AUTOMATIC GENERATION OF INTEGRATED LEXICAL ANALYSER-CUM-PARSERS WITH ERROR RECOVERY

by

V. H. SUBRAMANIAN

CS
1982
M
SUB
AUT

COMPUTER SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY, 1982

AUTOMATIC GENERATION OF INTEGRATED LEXICAL ANALYSER-CUM-PARSERS WITH ERROR RECOVERY

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

V. H. SUBRAMANIAN

to the

COMPUTER SCIENCE
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JULY, 1982

1 JUN 1984

CENTRAL LIBRARY
Kanpur.
Acc. No. A 82656

CS-1982-M-SUB-AUT

CERTIFICATE

This is to certify that the project entitled 'AUTOMATIC GENERATION OF INTEGRATED LEXICAL ANALYSER CUM PARSER WITH ERROR RECOVERY' has been carried out by Mr. V.H. Subramanian under my supervision and has not been submitted elsewhere for the award of a degree.

H.V. Sahasrabuddhe

Dr. H.V. Sahasrabuddhe
Head of the Department,
Computer Science Programme
Indian Institute of Technology, Kanpur
KANPUR-208016

Kanpur:
July 1982

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor H.V. Sahasrabuddhe for his inspiring guidance, constant encouragement and active involvement throughout the course of the work. His expert guidance has made this project a most pleasing and memorable experience.

I am thankful to all my friends for their invaluable suggestions. Special thanks are due to Mr. S.V.S. Nageswara Rao for his extensive participation in my thesis work and pleasant company to make my stay enjoyable.

Thanks are also due to Mr. M.C. Gupta for his excellent typing.

- V.H. Subramanian

Kanpur:
July'82:

CONTENTS

Chapter	Title	Page
I	Overview	1
II	Lexical Analyser Generator	4
III	Parser Generator	10
IV	Bootstrapping the Generators	15
V	Coelescing the Generators	18
VI	PLAG Input and Output	23
VII	Automatic Interfacing with Hand-Coded Semantic Processing	27
VIII	Epilogue and Suggestions for Further Work.	30
	References	
	Appendix	
	A. Example Specification and Generated Program	

ABSTRACT

A number of tools have been developed specifically to help construct compilers. These tools range from scanner and parser generators to complex systems, variously called compiler-compilers, compiler generators, or translator writing systems. In what we may call another significant step to reach the goal of writing a compiler-compiler, we coalesce the existing scanner generator and parser generator into an integrated system which generates a parser (with error recovery) together with a compatible lexical analyser from a description of the lexical and syntactic structure of the source language. Before the integration process we also bootstrap the existing lexical analyser generator and parser generator. Such a bootstrapping makes possible ready alteration of the specifications formats to be incorporated into the two generators without much effort. Also, we exploit the fact that the parser generated by the parser generator has a good error recovery which is now made available in the input phase of the generators.

The lexical analyser generator input is through regular expressions while the parser (LL(1) recursive descent) generator accepts an EBNF specification.

One significant advantage of using an integrated parser cum lexical analyser generator is increased reliability. An off-the-shelf, mechanically generated parser with lexical analyser is more likely to be correct than one produced by hand.

CHAPTER I

OVERVIEW

Our work is, basically, an extension to the parser generator (SD 81) and the Lexical Analyser Generator (SG 81). Presently, we have an LL(1) recursive descent parser generator which accepts a definition of the syntactic structure of the Language through EBNF and outputs a recursive descent parser with error recovery. The lexical analyser generator accepts an input specification through regular expressions. Both these generators scan through their input specification before the processing phase. Now, there are two major drawbacks in the input phase of these generators.

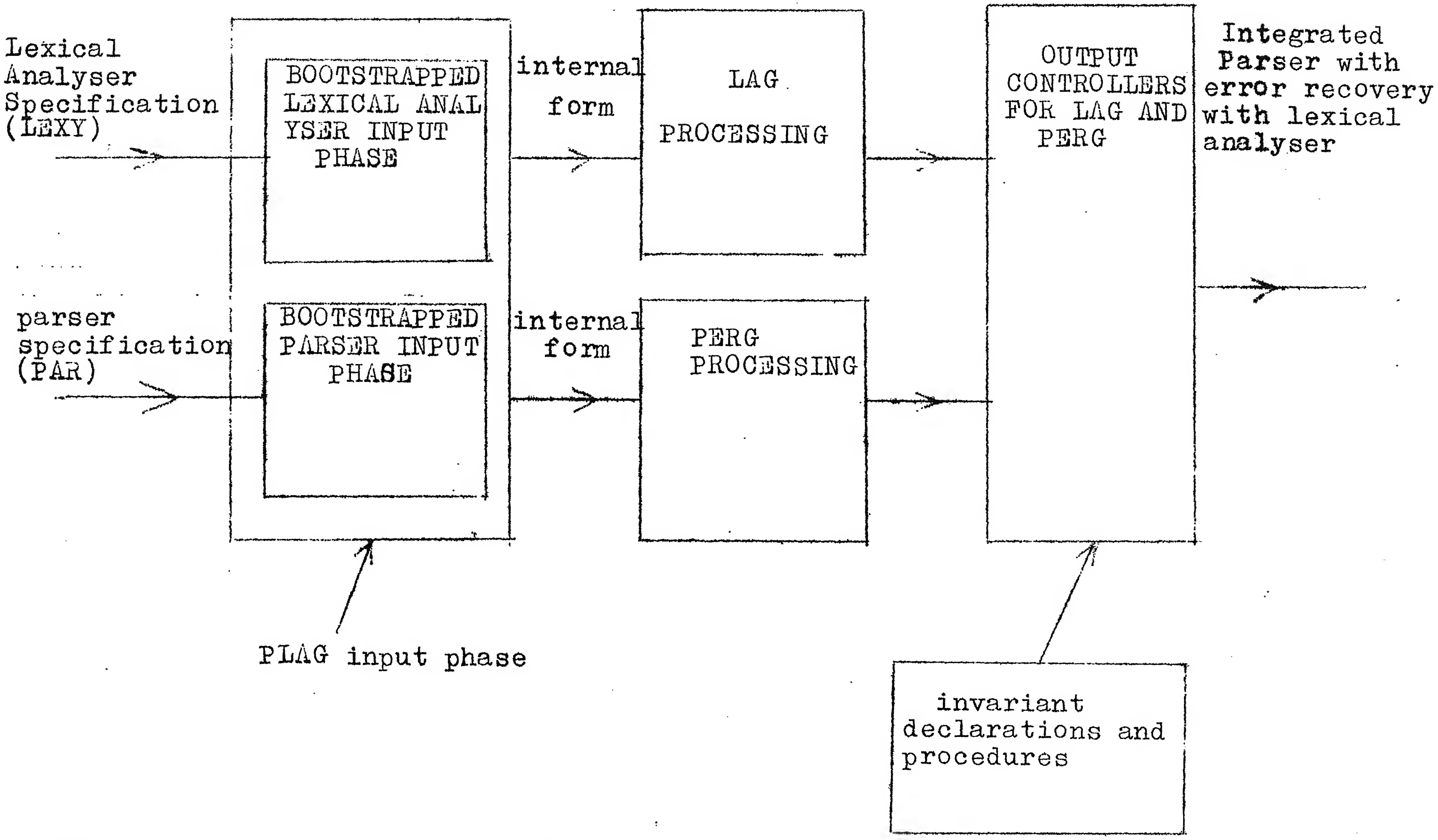
i) They do not have good error recovery and error reporting.

ii) Any slight ameliorations in the format of the specifications that the user may desire would mean a lot of patch work with the hand-coded scanners.

So, we obviate these problems for the user by bootstrapping the generators. That is to say, we replace the existing input phases of the generators by the integrated parser cum lexical analysers generated by these two generators for their input specifications. This, obviously, eliminates the problem of error recovery and reporting since the generated parser has both these features. Secondly, the user can, at his wish change the format of specifications through bootstrapping and some amount of hand-coding.

We have also made suitable changes in the lexical analyser generator's program synthesis phase so that the new lexical analyser is compatible with the generated parser. These two generators are coelesced into what we call a Parser with Lexical Analyser Generator (PLAG). PLAG (Fig. 1.1) makes use of some invariant declarations and procedures stored in other files. PLAG takes in the Lexical analyser specification (LEXY) and/or the parser specification (PAR) and generates a working parser with lexical analyser.

There is much convenience in this integration as the user does not have to bother about interfacing problems of the parser and lexical analyser.


Lexical
Analyser
Specification
(LEXY)
parser
specification
(PAR)
BOOTSTRAPPED LEXICAL ANALYSER INPUT PHASE
BOOTSTRAPPED PARSER INPUT PHASE
PLAG input phase
internal form
internal form
LAG PROCESSING
PERG PROCESSING
OUTPUT CONTROLLERS FOR LAG AND PERG
invariant declarations and procedures
Integrated Parser with error recovery with lexical analyser

LAG: Lexical Analyser Generator
PERG: Parser with Error Recovery Generator
PLAG: Parser with Lexical Analyser Generator

FIGURE 1.1 STRUCTURE OF PLAG

CHAPTER II

LEXICAL ANALYSER GENERATOR

In this chapter we briefly summarize the principal features of the Lexical Analyser Generator (LAG) (SG 81). The structure of LAG is shown in Figure 2.1.

2.1 The Generator:

2.1.1 Input Analyser: This phase is like the front end of a compiler. Lexemes are formally specified by regular expressions. The input analyser performs syntax analysis, with error recovery, of the specification. Further more, it constructs various internal tables for use by subsequent phases.

2.1.2 Internal Form Generation: The task of internal form generation is a sequence of 3 steps (Figure 2.2) concerning conversion of-

i) RE to non-deterministic finite automata (NFA).

ii) NFA to Deterministic finite automata (DFA).

iii) DFA to reduced DFA.

The output produced by this phase is a reduced DFA to process input characters for recognizing lexemes.

2.1.3 Lexical Analyser Program Sunthesizer:

The input to the synthesizer is a description of the following.

i) Transliteration table

ii) Minimized LEX DFA table

iii) Keyword wordlists

The task of the synthesizer is to produce an output program in s high level language whose code reflects the state transition structure of the generated lexical analyser.

2.2 Input Specification:

The input specification, essentially, consists of 4 declarations (Figure 2.3).

2.2.1 LITERAL Declaration:

Here the user defines the transliteration desired. He associates a subset of the ASC11 set with each identifier on the LHS. This subset may be specified by enumerating the individual characters in a string, or by defining a range with a lower and an upper bound. This transliteration permits even overlapping subsets on the RHS. The ambiguity is resolved by mapping a character on to the literal it was last defined under.

```
e.g.      LETTER = 'A' . . 'Z' ;
          EXP    = 'E'
```

'E' will map onto EXP and not to LETTER.

Finally, the use of the keyword SKIP on the LHS defines a special literal. Characters which map onto this literal are simply elided from the input stream.

2.2.2 TOKEN Declaration:

The identifiers on the LHS represent the tokens which the generated lexical analyser will recognize. They are defined as regular expressions over literal identifiers. The Keyword NULL may also be used in the RHS to represent the empty string.

```
e.g.      IDENT  = LETTER (LETTER ¦DIGIT)  ;
          NUMBER = DIGIT + ;
          A      = DOT (DOT: NULL)
```

2.2.3 DELIMITER Declaration:

This declaration is optional. A list of literal identifiers is specified. These literals are treated as delimiters between tokens. In effect, the generated lexical analyser will skip over an initial sequence of such literals before beginning to accept a token.

e.g.
```
DELIMITER   BLANK
```

2.2.4 KEYWORD Declaration:

The lexical analyser generated by LAG employs the 'reserved word strategy' (AU 77), a method which is frequently used in practice. This technique associates reserved words with a token. When the lexical analyser recognizes this token, the substring accepted is compared with the reserved words. If a match is found, the Lexeme returned is the one corresponding to the Keyword matched. If not, the Lexeme is the token recognized.

The user can associate keywords with tokens through this declaration. This is done by enumerating a list of string-identifier pairs. In a pair, the string is the keyword and the identifer is the Lexeme associated with it. We call such a list a 'word list'.

e.g.
```
TOKEN IDENT
WL  =<'BEGIN' BEGINSY  ,  'END' ENDSY>
TOKEN NUM
CRYPTIC = <'007' JB>
```

We postpone the discussion on the output of the LAG to Chapter IV.

-:o0o:-

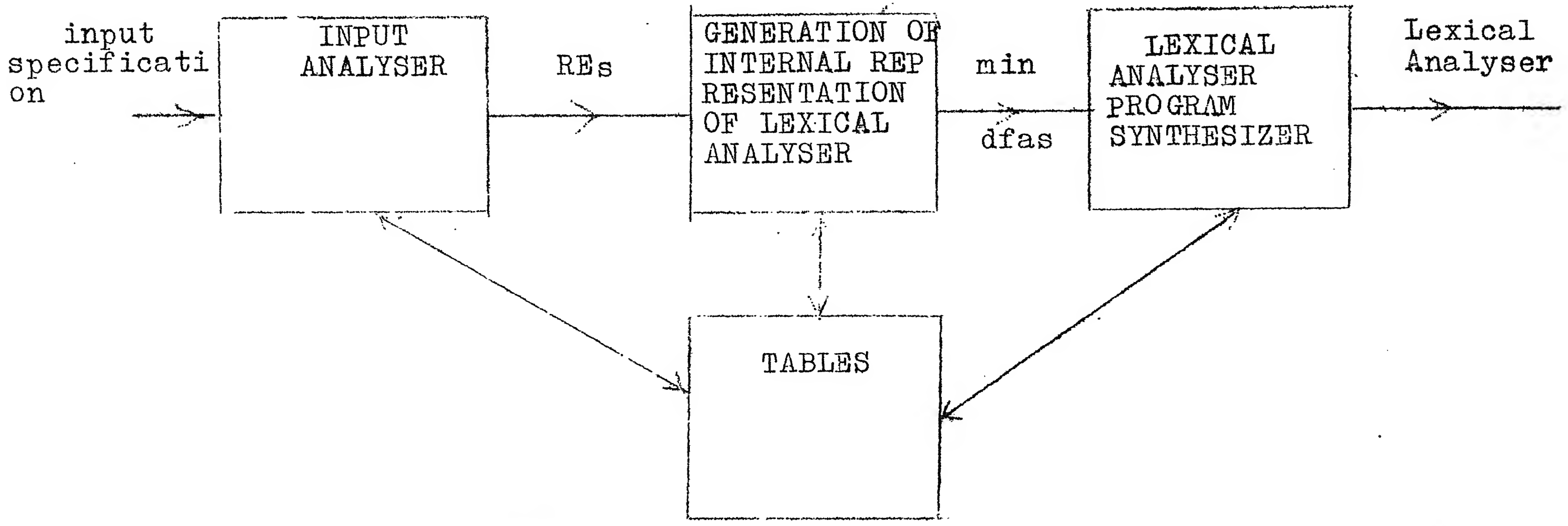

input specificati on
INPUT ANALYSER
REs
GENERATION OF INTERNAL REP RESENTATION OF LEXICAL ANALYSER
min dfas
LEXICAL ANALYSER PROGRAM SYNTHESIZER
Lexical Analyser
TABLES

FIGURE 2.1 LEXICAL ANALYSER GENERATOR (LAG)

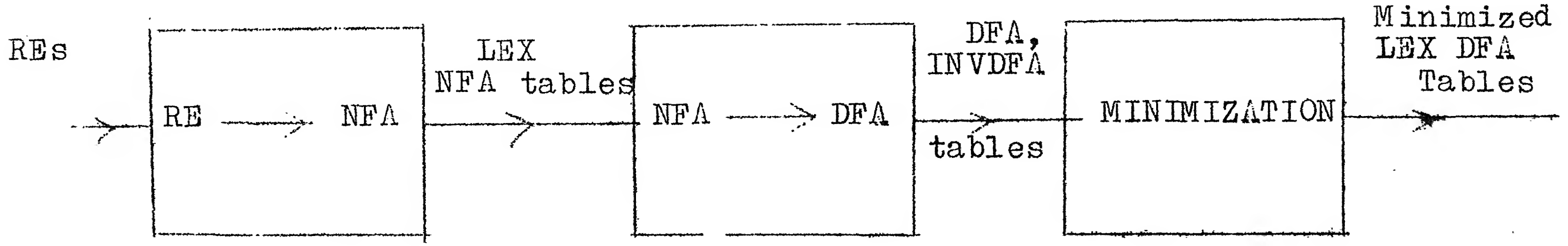

REs
RE
NFA
LEX NFA tables
NFA
DFA
DFA, INVDFA tables
MINIMIZATION
Minimized LEX DFA Tables

FIGURE 2.2 INTERNAL REPRESENTATION GENERATOR

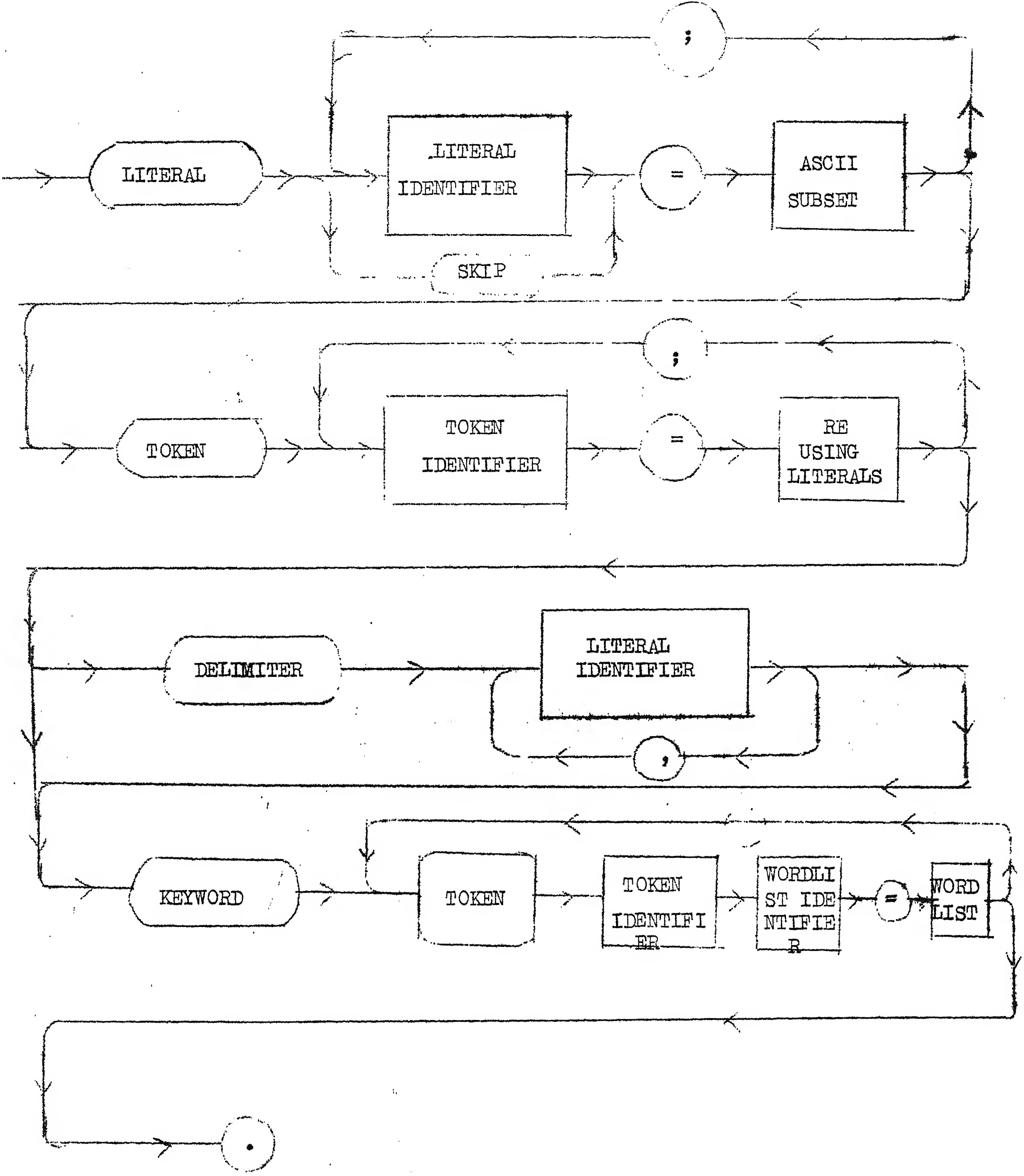

;
LITERAL
LITERAL IDENTIFIER
=
ASCII SUBSET
SKIP
;
TOKEN
TOKEN IDENTIFIER
=
RE USING LITERALS
DELIMITER
LITERAL IDENTIFIER
,
KEYWORD
TOKEN
TOKEN IDENTIFIER
WORDLIST IDENTIFIER
=
WORD LIST
.

FIGURE 2.3

Syntax tree for LAG input specification

CHAPTER III

PARSER GENERATOR

This chapter deals with the method of generation of recursive descent parsers with error recovery for LL(1) grammars from the specification of the grammar in EBNF (SD 81). The structure of PERG is shown in Figure 3.1.

3.1 The Generator

3.1.1 Input Analyser:

The input analyser performs syntax analysis and context-sensitive analysis, with error recovery on the input specification. It also constructs the various internal tables. The following context-sensitive checks are made on the specification.

1. The set of terminal and non-terminal symbols should be disjoint.
2. Any terminal or non-terminal symbol should not be declared more than once.
3. The use of a terminal or non-terminal symbol in a production must be preceded by its declaration.
4. There should be no more than one production for the same non-terminal.
5. There must be a production for the axiom.
6. All the non-terminals accessible from the axiom should have expansions.

Error reporting is carried out if any of the conditions is violated.

3.1.2 Internal Form Generation:

The following tables are generated by the input analyser.

1. Two tables for storing names of terminal and non-terminal symbols in the alphabetic order. A terminal or a non-terminal is identified by its position in the table.
2. ACCESSSET (array of sets), to contain the set of non-terminals accessible from a non-terminal. AXACCSET denotes the set of non-terminals accessible from the axiom.
3. DELIMSET, to contain the set of MLM (most likely to be missing) symbols in the grammer.
4. PRODARRAY, to store the productions of the grammar in sequence.

3.1.3 Grammar Processor:

The structure of the grammar processor is shown in Figure 3.2.

3.2 Error Recovery:

The error recovery in the generated parser is the one introduced by PAI (PK 80). We have a two-level recovery strategy-local and global. During error-recovery, a basic consideration is that the erroneous text closely approximates some correct sentence of the language. Hence the error-recovery scheme should first attempt to correct the text locally around the point of error so as to get a legal sentential prefix. The actions may involve the insertion, replacement or deletion of a certain number of symbols. Only if local recovery fails, we should use the global recovery strategy.

a) LOCAL: In this phase, a single-token correction is attempted at the point of the error. Insertion and then replacement of a single token is attempted, while the responsibility for deletion is deferred to the global recovery phase. We require the lexical analyser to handle two Lexeme look-ahead.

b) GLOBAL: If the local recovery phase fails to take definite and unambiguous decision, control passes to the global recovery scheme, that works in 'panic-mode' (Gri 76). In this phase the input is scanned until one of a set of recovery symbols is encountered. The skelton of this scheme is based on Amman's scheme (Amm 78).

In LL(1) parsing, an error is discovered when the current look-ahead symbol does not match the expected terminal symbol generated by a left-most canonical derivation. Therefore, whenever there is a definite expectation of a terminal symbol, or a set of terminal symbols at the current point of parse, a procedure TESTSYS is called that discovers the error, attempts Local repair, failing which it performs global recovery. TESTSYS is aided in this process by the sets of terminal symbols, that are characteristic of the CFG for the purposes of error detection, local repair and global recovery.

3.3 Input Specification:

The syntax of the input specification is given in Figure ~~3~~4.3.

In the specification the only value of selident allowed is 'D' and its presence implies that this tsym should be treated as an MLM symbol.

-:oOo:-

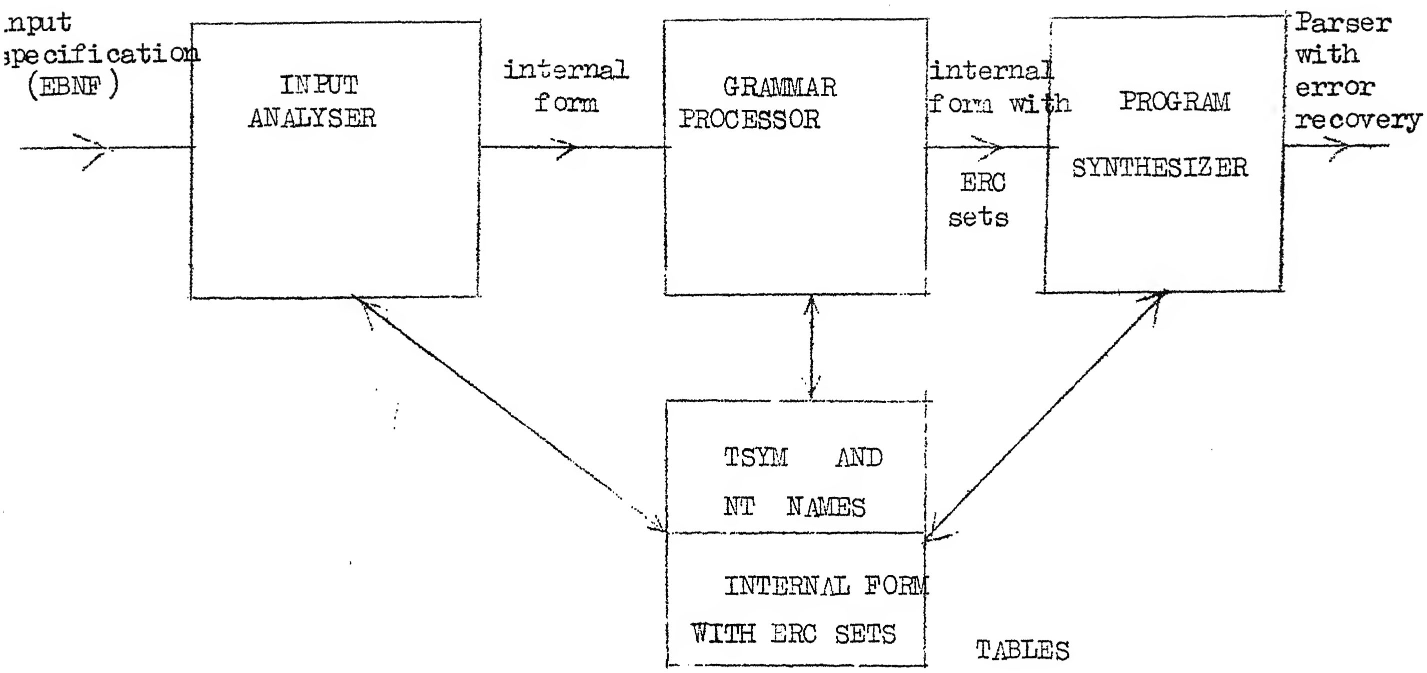

nput
pecification
(EBNF)
INPUT
ANALYSER
internal
form
GRAMMAR
PROCESSOR
internal
form with
ERC
sets
PROGRAM
SYNTHESIZER
Parser
with
error
recovery
TSYM AND
NT NAMES
INTERNAL FORM
WITH ERC SETS
TABLES

FIGURE 3.1

STRUCTURE OF THE PERG SYSTEM

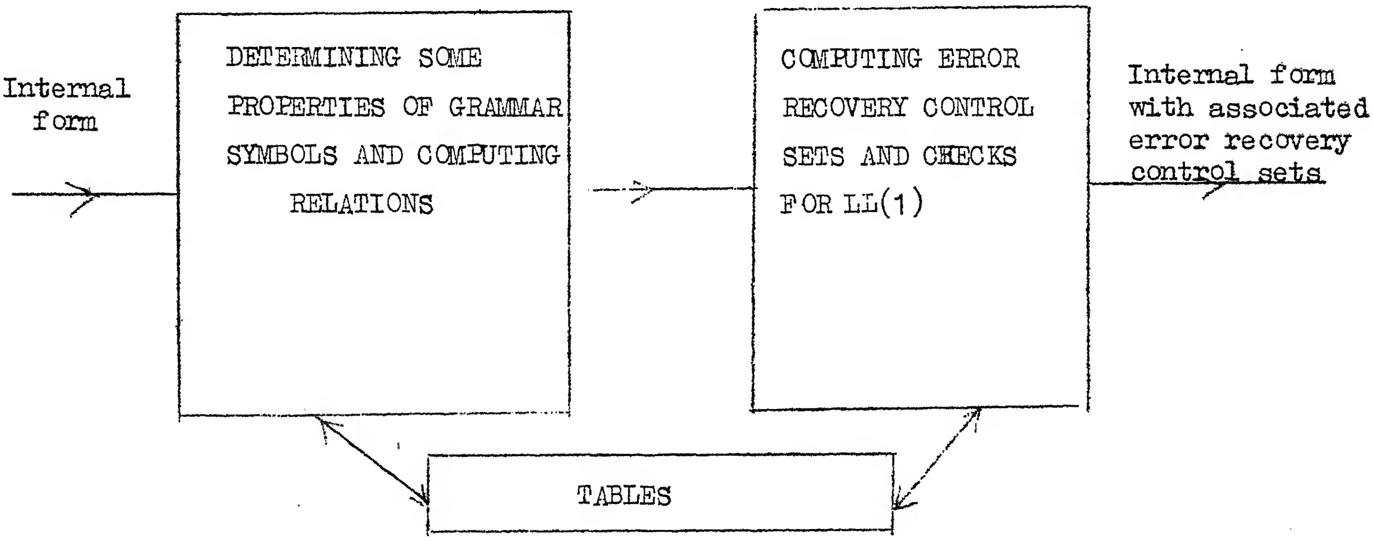

Internal
form
DETERMINING SOME
PROPERTIES OF GRAMMAR
SYMBOLS AND COMPUTING
RELATIONS
COMPUTING ERROR
RECOVERY CONTROL
SETS AND CHECKS
FOR LL(1)
Internal form
with associated
error recovery
control sets
TABLES

FIGURE 3.2

GRAMMAR PROCESSOR

CHAPTER IV

BOOTSTRAPPING THE GENERATORS

In this chapter, we devote to the need for and the advantages gained in bootstrapping the input phases of the two generators. As explained in chapter II and III the LAG and PERG generators had originally hand-coded lexical analysers for their input specifications. Also, these lexical analysers did not have good error recovery features. So the user, often, had problems in coming out with the correct specification when he wanted to generate lexical analysers and/or parsers. In the case of PERG which had no error recovery at all he had to run the program not less than 3 to 4 times which proved to be extremely wasteful. In the case of LAG, the user had problems trying to decipher the errors from the cryptic error report produced. So much so, the original versions of LAG and PERG taxed the user in stipulating him to come up with the right specification. We have obviated this difficulty by replacing the old input analysers with parsers with good error recovery through bootstrapping.

One more consideration which made us effect this change was the fact that the user should be able to alter the input specifications to suit his needs without much effort. That is to say, by making the least number of changes in the input phase. We achieve this again through bootstrapping. Now, what the user needs to do is whenever he wants to change the format of the input specification, he replaces the existing input phases by the generated input phases for the new specifications. After this he'll have to make some minor changes like insertion of some hand-coded procedures or other code to carry the necessary actions when specific lexames are read.

Lastly whenever the user discovers errors in the input phase he can just bootstrap to replace the erroneous version.

4.1 Bootstrapping the Generators:

4.1.1 Bootstrapping the LAG:

We write a specification for the LAG to generate a lexical analyser for its own input specification (see Fig. 4.1). Likewise we write another specification for the PERG to generate a parser with error recovery for the input specification of LAG (see Fig. 4.2). Or in the coelesced version (PLAG) we write lexical analyser and parser specifications for the LAG input and generate the parser (with error recovery) integrated with lexical analyser (~~Program 4.1~~). We use this for scanning the input specification of LAG and checking for syntax. (instead of the existing input phase)

4.1.2 Bootstrapping the PERG:

This is identical to the bootstrapping of LAG. Here we write lexical analyser and parser specification for the PERG input and generate a parser (with error recovery) integrated with lexical analyser for the PERG input (~~Prog.4.2~~). See Figure 4.3 , 4.4.

We replace the existing hand-coded version by this bootstrapped version.

4.1.3 Hand-Coding:

After generating the parsers for the specifications we introduce some code either in the form of procedures or otherwise mostly at the beginning of every non-terminal procedure. This is necessary as some actions are taken where specific tokens are recognized- for instance updating tables, lists, construction of nfas etc. It is here that the effect of bootstrapping is not rendered to the fullest extent but it could not be helped with the present versions of LAG and PERG.

4.2 Handling a new specification:

If the user wants to make some changes in the specifications formats all he has to do is write the lexical analyser and parser specification for the new format and generate the parser with lexical analyser. Next he replaces the existing parser with the new one.

Finally the user has to insert some code for every non-terminal procedure so that the replaced version conforms with the original one.

CHAPTER V

COELESCING THE GENERATORS

5.1 An Overview of the Outputs of LAG and PERG:

5.1.1 Lexical Analyser:

The LAG takes in a specification of the lexicon in the form of regular expressions. It outputs a collection of procedures which could be called to get the next lexeme in the input stream. In what follows we describe in minor detail as to the nature of these procedures.

i) Nxtlit: This is just like the character processing routine in a compiler. It makes use of the LIT array which contains the literal values associated with all characters in the ASCII set. The parameter passed on by this routine is just the literal value associated with the next character in the input stream.

ii) Initialise: Basically an initialization routine initialise performs the character-literal association for the entire ASCII. It establishes the keyword record information for all the keywords declared by the user and maintains the list of tokens which have keywords associated with them. Finally it sets up information of the literals which are to be treated as delimiters in the input stream.

iii) Nxtsym: Nxtsym supplies the lexical value of the next lexeme encountered in the input stream (every token and keyword in the lexicon defined by the user is assigned a unique lexical value). It is basically a series of goto statements embedded with calls to Nxtlit and store (to keep track of the token during the process of calls to Nxtlit). Finally, on encountering tokens which have keywords it does a binary search on the appropriate keyword list. On success the lexical value associated with the keyword string supercedes the earlier value.

LINE is an array containing the current line of the input stream while BUF stores the current lexeme that is being scanned.

5.1.2 Parser:

The user specifies the syntax in EBNF. PERG generates an LL(1) recursive descent parser with error recovery. The specification consists of the set of productions, the set of terminal (lexemes) and non-terminal symbols and the goal symbol. Each non-terminal that the user defines appears as a procedure in the parser generated. Apart from this there are other error routines, accept routines and skip routines which are invariant.

Error, Errormessage, Processerror,

Lexerror, errorset, errorsym, skiperror, localerror:

These are the invariant error routines outputted by PERG. Calls to these routines are made whenever an error is encountered in the syntax of the input.

TESTSYS: Whenever an error in the syntax is encountered in the input stream this routine ensures that the succeding lexemes in the stream are skipped till a parsable point is reached. Calls to Testsys are embedded in every non-terminal procedure.

INITPREVSETS: It provides information essential to this error recovery process. It is, basically, a list of sets of all symbols which could precede each terminal symbol in the input stream.

LEXANALYSE: Provides the next lexeme (of type enumeration) by calling the lexical analyser.

5.2 Incompatibility of the Lexical Analyser and the Parser:

In its original version the lexical analyser was just supplying a lexical value of the lexeme encountered while the parser assumed interfacing with a hypothetical lexical analyser which supplied an enumeration type of the next lexeme encountered. To get by this discrepency we had to modify the lexical analyser to pass on the next lexeme of type enumeration.

To aid this we now have an initialisation routine called Initsypos which associates to all the lexical values of the lexemes. Their corresponding enumeration type names. We also have procedure Initsymnames which initialises the token and keyword arrays and is used by Errormessage. Nxtlit had to be modified to take care of the processing of the errors in the previous line whenever eoln is encountered. Procedure Lexanalyse of the parser also underwent some changes so as to interface properly with the lexical analyser.

5.3 The Coelesced Generator PLAG:

Figure 1.1 shows PLAG in its present version. The user can generate either or both the lexical analyser and the parser. LEXY takes in the lexical analyser specification while PAR takes in the parser specification. On providing both LEXY and PAR the user can generate an integrated lexical analyser-cum-parser with error recovery. PLAG makes use of some invariant routines like the character processing routine (nxtlit), error routines and other variable declaration is INVDEC and NXTLIT. The output controllers for LAG and PERG in the PLAG system ensure proper outputting of the various procedures. Specifically, the output controller phase of LAG underwent lot of changes to ensure generation of a lexical analyser that is compatible with the parser. Many new procedure have been added in PLAG to this effect. The existing input analysers in the LAG and PERG phases have been replaced by the bootstrapped versions as explained in chapter IV.

5.4 Efficiency Considerations of the Parser-cum-Lexical Analyser:

One of the primary drawbacks of programs generated by systems like PLAG is that they are not as efficient as ones written by hand. But we bank on the fact that generated programs are much more reliable than hand-coded ones. The chief problem is that there is a trade-off between how much work the generator system can do automatically for its user and how flexible the system can be. We have tested the programs outputted by PLAG and then results have been very encouraging. For instance, the parser-cum-lexical analyser for the language pascal has compared reasonably well with its hand-coded counterpart PASREL.

CHAPTER VI

PLAG INPUT AND OUTPUT

6.1 Input Specifications:

The input specifications to the PLAG are explained in chapters II and III. We have generated a Parser-cum-lexical analyser for PASCAL (with certain limitations) as Pascal does not enjoy the LL(1) property. Figure Nos. 6.1, 6.2 show the lexicon and parser specification of the language.

6.2 PLAG Output:

Most of the details of the program generated by PLAG have been discussed in chapter V. The parser-cum-lexical analyser generated for pascal is listed in Program 6.1.

6.3 Critical Appraisal of Pascal Syntax:-

Pascal grammar does not enjoy the LL(1) property because of the dangling else problem. Binding the else clause to the closest if-then construct is a programatic solution of the problem. However, we obviate this short fall by generating a pascal syntax analyser that caters to only the if-then construct and a lexical analyser that recognizes the else symbol. After generating we modify the non-terminal IFSTMT procedure so as to continue parsing whenever an else symbol is encountered.

This is done as follows. Else-sy being a keyword that is declared, we don't skip elsesy when encountered in the input stream (see the modification in procedure Testsysnew Figure 6.3). Also, when an elsesy is recognized after the if condition then statement construct in the input code it is just globbed up and the next symbol is requested. On meeting an if symbol procedure ifstmt is called again else procedure statement is called (see Figure 6.4 and 6.5). Thus, by introducing some amount of hand-code we obviate the non-LL(1) nature of Pascal syntax.

Apart from this, Pascal syntax has the following drawbacks:

1. In type-denoter, both simple type and enumeration type may start with an identifier, arbitration is possible only after the scanning of the next symbol. Due to this delayed arbitration, syntax can not reflect the semantics property.

2. Writing LL(1) grammar for the optional semicolon before the 'end' of the record declaration and case statement construct is quite involved and the grammer becomes messy.

Error Recovery Property:

The factor most detrimental to the error recovery scheme is the overloading of 'END' and 'BEGIN' symbols. Due to their dual roles and boundary symbols for block and statement, any mistake concerning these symbols will cause not only a misinterpretation about the body of the current block, but it is highly probable that the effect will be propogated resulting in misinterpretation about the body of other blocks also. Since the important function of context switching takes place at block boundaries, this will create a long stream of impleasant sympathetic error message. One solution to such a problem could be the use of'blockbegin' and 'blockend' as end symbols for a block, such that proper error recovery actions can be taken based only on syntax.

Switches:

Two switches C and F are provided in the parser specification. The violation of LL(1) property does not stop the process of code generation if C switch is off; whereas if it is on, the code generation process is blocked if the grammar is not LL(1). The status of the forward switch dictates whether forward declarations would be provided (F switch is on) or not (F switch is off). By default C is on and F is off.

Error Messages:

As and when errors in syntax are encountered in the PLAG specifications they get displayed on the TTY. Due to the very good error recovery scheme that has been provided at the input analyser phases of PLAG, it becomes very convenient to the user to come up with error-free specifications without much effort.

-:o0o:-

CHAPTER VII

AUTOMATIC INTERFACING WITH HAND-CODED SEMANTIC PROCESSING

We have, now, an automated aid for generating a lexical analyser-cum-parser available. It seems appropriate at this stage to investigate the interfacing of proper semantic processing and code generation with the present lexical analyser-cum-parser. In this chapter we suggest a way of automatic interfacing of this output phase with the rest of the phases which we assume will be hand-coded.

Interfacing with the Outside World:

Every non-terminal in the parser grammar specification appears in the generated parser as a procedure definition. Each such procedure consists of:-

1. Calls to the error recovery routine TESTSYS
2. Calls to non-terminal procedures if appropriate points (including itself)
3. Accepting or gobbling up of terminal symbols **whenever** the syntax stipulates.

Almost all of the semantic processing could be done at the beginning/end of a procedure definition and before/after any terminal symbol is gobbled up. So we suggest an automatic means of spewing out every non-terminal procedure in the parser generated with Hand procedure calls at appropriate places. This would mean that, at all these places where we have calls to Hand procedures we could do semantic processing by defining the same in the outside world. Whenever we decide not to do any processing at a particular Hand procedure, we can define a null procedure (a procedure which does nothing) in the outside world.

The success of such approach will be ensured if we can automatically cook unique names for each of such Hand procedures when the parser is generated. The need for such a uniqueness will be clear in the following treatment.

1. The first statement of any procedure after the call to Testsys will be a Hand Procedure call of the form 'Handprocname' where procname is a string of characters preferably the first few and sufficiently long to uniquely identify it from other such Hand procedures. The names of these Hand procedures have to be distinct as otherwise will land up with an unpleasant and illegal situation of having two Hand procedure definitions with the same name at the same level in the outside world.

2. Whenever a procedure is generated we set a counter variable. To ensure uniqueness of Hand procedures within that procedure from every other Hand procedure other than the first within that procedure we append the counter value to the string of characters standing for that procedure name and

increment the counter for every such Hand procedure call generated. There are 3 places at which there secondary Hand procedure calls are generated:-

a). After every Accept statement

b). After the beginsy following every 'while Chksymset ([S1, S2..])' construct

c). After the thensy following every 'if Chksymset ([S1,..]) construct

3. We can also introduce special marker symbols in the specification to dictate generation of Hand procedure wherever they are tagged. For instance at the beginning of a production the user can insert a marker symbol so that in the parser generated the procedure corresponding to that non-terminal starts off with a Hand procedure call. This gives the fullest flexibility for Hand procedure generation.

By this we now have a systematic way of cooking Hand procedure names. Once this is done the interfacing with the outside world becomes very easy. A switch could be provided in the specification to suppress such a generation of Hand procedures when the user wants.

PROG 7.1 shows how a sample non-terminal procedure would look like after spewing out hand procedures at suitable points.

CHAPTER VIII

EPILOGUE AND SUGGESTIONS FOR FURTHER WORK

With an automated aid for generating a parser-cum-lexical analyser we are now one more step ahead towards our goal of developing a compiler compiler. This experience has been quite rewarding.

Scope for improvement in the present version of PLAG lies in:-

1. The choice of efficient data structure for representing the productions. The use of linked list structure, that stores the parse tree for the productions could make the algorithm more efficient, since it allows proper association of control sets to be used by successive passes and avoids the need for rebuilding the parse tree during every pass of the grammar. The dynamic space allocation also removes the arbitrary limit on the total length of productions.

2. The use of an abstract data type BIGSET (as in LAG) with operations defined on it rather than the standard set in Pascal. This will increase the upper bound on the number of terminals and non-terminals in the input grammar.

The possible extensions to the PLAG are:-

1. The automatic generation of nesting structure for the parser. Right now all the procedures are produced at the same level, augmented by necessary forward declarations

(when F switch is on). A better approach would be the generation of a properly nested parser.

2. The modifications to handle L-attribute grammars (Kch 81, Sgh 81).

3. Development of automated aids for semantic processing and code generation.

REFERENCES:

1. Amm 78 Amman, U.: Error Recovery in Recursive Descent Parsers. Tech. Rep. 25, Institut fur informatik, ETH Zurich, May 78.

2. AU 77 Aho, A.V., Ulman, J.D.: Principles of Compiler Design. Addison-Wesley Publishing Company, 1977.

3. SD 81 Datta, S.: Generation of LL(1) recursive descent parsers with error recovery from EBNF specifications.

4. Dey 80 Dey, A.K.: Language Processors: An Exercise in Systematic Program Development. M.Tech. Thesis, IIT Kanpur, 1980.

5. Gri 76 Gries D.: Error Recovery and Correction - An Introduction to the Literature. In Compiler Construction : An Advanced Course, Ed. by Bauer, F.L., Eickel, J., Springer--Verlag, NY 1976.

6. JW 75 Jenson, K., Wirth N.: PASCAL - User Manual and Report, Springer Study Edition, 1975.

7. Kch 81 Kachhwaha, P.: Synthesis of Static Context in PASCAL Programs with L-Attribute Grammars. M.Tech. Thesis, IIT Kanpur, 1981.

8. SG 81 Sarkar, V., Gupta, R.: A Command Language Processor Generator. B.Tech. Thesis, IIT Kanpur 1981.

9. Sgh 81 Singh K.: Static Contextual Analysis of PASCAL Programs with L-Attribute Grammars. M.Tech. Thesis, IIT Kanpur, 1981.

CENTRAL LIBRARY
I. I. T., Kanpur.
No. A 82656

```
LITERAL
A = '=,;|*+()<>:' , ;
STOP =          '.' ;
LETTER = 'A' .. 'Z' ;
DIGIT = '0' .. '9' ;
QUOTE = '''' ;
BLANK = ' '

TOKEN
TDOT = STOP  ;
ONECH = A ;
TDOTDOT = STOP STOP  ;
TIDENT = LETTER ( LETTER | DIGIT ) *  ;
STRING = QUOTE ( A | STOP | BLANK | LETTER | DIGIT | QUOTE QUOTE ) * QUOTE

DELIMITER
BLANK

KEYWORD
TOKEN
WL1 = <'=' TEQ><',' TCOMMA>,<';' SEMICOL>,<'|' VBAR>,<'*' STARSY>,
       <'+' PLUSSY>,('(' TLPAR>,<')' TRPAR>,<'<' LBRAC>,<'>' RBRAC>,
       <':' TCOLON>
TOKEN  TIDENT
WL2 = <'LITERAL' LITERALSY>,<'SKIP' SKIPSY>,<'TOKEN' TOKENSY>,
<'DELIMITER' DELIMSY>,<'KEYWORD' KEYWORDSY> .
```

fig 4.1. Lexicon specification of LAG input

```
[C+F+]
( < CLPSPEC , LTIERALDEC , TOKENDECL , DELIMITERD , KEWORDDEC , LTIDEF ,
    UNIT , RANGE , REGULAR , REGDEF , RE , TERM , FACTOR , ONEDELIM ,
    TOKENWLDEC , WLDEF , PAIR , PAIRLIST >
  ,
  < TDOT , TDOTDOT , TIDENT , STRTNG , NULLSY , TEQ(D) , TCOMMA(D) ,
    SEMICOL(D) , VBAR(D) , STARSY , PLUSSY , TLPAR(D) , TRPAR(D) ,
    LBRAC(D) , RBRAC(D) , TCOLON(D) , LTIERALSY , SKIPSY , TOKENSY ,
    DELIMSY , KEYWORDSY >
  ,
  < CLPSPEC      --> LTIERALDEC TOKENDECL [DELIMITERD] [KEYWORDDEC] 'TDOT' ,
    LTIERALDEC   --> 'LITERALSY' LTIDEF { 'SEMICOL' LITDEF } ,
    LTIDEF       --> 'TIDENT' 'TEQ' UNIT ,

    UNIT         --> 'STRING' [RANGE] ,
    RANGE        --> 'TDOTDOT' 'STRING' ,
    TOKENDECL    --> 'TOKENSY' REGULAR ,
    REGULAR      --> REGDEF { 'SEMICOL' REGDEF } ,
    REGDEF       --> 'TIDENT' 'TEQ' RE ,
    RE           --> TERM { 'VBAR' TERM } ,
    TERM         --> FACTOR { FACTOR } ,
    FACTOR       --> ( 'TIDENT'/'TLPAR' RE 'TRPAR'/'NULLSY' ) [ STARSY/PLUSSY] ,,
    DELIMITERD   --> 'DELIMSY' ONEDELIM { 'TCOMMA' ONEDELIM } ,
    ONEDELIM     --> 'TIDENT' ,
    KEYWORDDEC   --> 'KEYWORSY' { 'TOKENSY' TOKENWLDEC } ,
    TOKENWLDEC   --> 'TIDENT' WLDEF ,
    WLDEF        --> 'EQ' PAIRLIST ,
    PAIRLIST     --> PAIR { 'TCOMMA' PAIR } ,
    PAIR         --> 'LBRAC' 'STRING' 'TIDENT' 'RBRAC'  >
  ,
    CLPSPEC  )
```

fig. 4.2. Syntax specification of LAG

```
LITERAL
LETTER          =          'A'..'Z';
D               =          'D';
DIGIT           =          '0'..'9';
QUOTE           =          '''';
LPAR            =          '(';
RPAR            =          ')';
GTS             =          '>';
HYPHEN          =          '-';
OTHER           =          '[]{},/+<';
BLANK           =          ' '

TOKEN
IDENT           =          (LETTER|D) (LETTER|D|DIGIT)* (LPAR D RPAR | NULL );
ITDENT          =          QUOTE (LETTER|D) (LETTER|D|DIGIT)* QUOTE;
ARROW           =          HYPHEN HYPHEN GTS;
LPAREN          =          LPAR;
RPAREN          =          RPAR;
OTHERTOK        =          OTHER;
GT              =          GTS

DELIMITER
BLANK

KEYWORD
TOKEN OTHERTOK
WL1       =        <'[' LBRAC>,<']' RBRAC>,<'{' LCBRAC>,<'}' RCBRAC>,
                   <',' COMA>,<'/' SLASH>,<'+' PLUS>,<'<' LT> .
```

fig 4.3 Lexicon specification of PERG inpu

```
[C+F+]
(    < PERGSPEC , TSYMLIST , NTSYMLIST , TSYMNAME , NTSYMNAME , PRODLIST ,
   EXP ,PROD   , TERM , FACTOR , TSYM , NTSYM , AXIOM >
,
  <  LPAREN(D) , RPAREN(D) , LBRAC , RBRAC , LCBRAC , RCBRAC , LT(D) ,
    GT(D) , COMA(D) , ARROW(D) , SLASH , TIDENT , IDENT >
,
  < PERGSPEC    -->   'LPAREN' 'LT' TSYMLIST 'GT' 'COMA' 'LT' NTSYMLIST 'GT' 'COMA'
   'LT' PRODLIST 'GT' 'COMA' AXIOM 'RPAREN' ,
    TSYMLIST    -->   TSYMNAME { 'COMA' TSYMNAME } ,
    NTSYMLIST   -->   NTSYMNAME { 'COMA' NTSYMNAME } ,
    TSYMNAME    -->   'IDENT' ,
    NTSYMNAME   -->   'IDENT' ,
    PRODLIST    -->   PROD { 'COMA' PROD } ,
    PROD        -->   NTSYM  'ARROW' EXP ,
    EXP         -->   TERM { 'SLASH' TERM } ,
    TERM        -->   FACTOR { FACTOR } ,
    FACTOR      -->   TSYM / NTSYM / 'LPAREN' EXP 'RPAREN' / 'LBRAC' EXP 'RBRAC' /
   'LCBRAC' EXP 'RCBRAC' ,
    TSYM        -->   'TIDENT' ,
    NTSYM       -->   'IDENT' ,
    AXIOM       -->   'IDENT' >
,
  PERGSPEC )
```

Fig 4.4. Syntaax specification of PERG input.

```
procedure TESTSYS ;
    begin
        ACCSYS:=ACCSYS+[ELSESY];
        if (not (SYM in ACCSYS))  then
          begin
              TOTSYS:=ACCSYS+STOPSYS;
              if  (RECOVERY<>NONLOCAL) and (not ATIMPTRECV) then
                begin
                    S:=ACCSYS*PREVSET[SYM];
                    if CARD(S)<=1 then
                      begin
                          if (CARD(S)=1)  then
                            begin
                                  begin PRESERVENExtsym;ATIMPTRECV:=true;SYM:=ELMT(S);
                                      RECOVERY:=LOCAL;      LOCALERROR(SYM,INSERTION);
                                  end
                            end
                          else
                                if (not (SYM in (TOTSYS))) then
                                  begin
                                      PRESERVESYm;LEXANALYSE;ATIMPTRECV:=true;
                                      PRESERVENExtsym;RESTORESYM;S:=ACCSYS*PREVSET[NEXTSYM];
                                      if (CARD(S)=1)
                                      then
                                        begin SYM:=ELMT(S);
                                            RECOVERY:=LOCAL;LOCALERROR(SYM,REPLACEMENt);
                                        end
                                      else SKIPSYS;
                                  end;
                      end
                    else SKIPSYS;
                end
              else SKIPSYS;
          end;
    end;
```

fig. 6.3 TESTSYSNEW .

```
procedure TESTSYS ;
  begin
    ACCSYS:=ACCSYS+[ELSESY];
    if (not (SYM in ACCSYS))  then
     begin
       TOTSYS:=ACCSYS+STOPSYS;
       if  (RECOVERY<>NONLOCAL) and (not ATIMPTRECV) then
        begin
          S:=ACCSYS*PREVSET[SYM];
          if CARD(S)<=1 then
           begin
             if (CARD(S)=1)  then
              begin
                 begin PRESERVENExtsym;ATIMPTRECV:=true;SYM:=ELMT(S);
                   RECOVERY:=LOCAL;    LOCALERROR(SYM,INSERTION);
                 end
              end
             else
                if (not (SYM in (TOTSYS))) then
                 begin
                   PRESERVESYm;LEXANALYSE;ATTMPTRECV:=true;
                   PRESERVENExtsym;RESTORESYM;S:=ACCSYS*PREVSET[NEXTSYM];
                   if (CARD(S)=1)
                   then
                    begin SYM:=ELMT(S);
                      RECOVERY:=LOCAL;LOCALERROR(SYM,REPLACEMENt);
                    end
                   else SKIPSYS;
                 end;
           end
          else SKIPSYS;
        end
       else SKIPSYS;
     end;
  end;
```

fig. 6.3 TESTSYSNEW .

```
procedure IFSTMT;
        begin
        TESTSYS([IFSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST

        REPEATSY,SIGN    ,STRGCONST,THENSY,WHILESY,WITHSY]+FSYS);
        ACCEPT(IFSY);
        EXPRESSION([THENSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,THENSY,WHILESY,WITHSY

        FSYS);;
        ACCEPT(THENSY);
        STMT(ACCFSYS,FSYS);
        end;
```

fig. 6.4 generated version of IFSTMT .

```
procedure IFSTMT;
     label 4 ;
        begin
        TESTSYS([IFSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST

        REPEATSY,SIGN    ,STRGCONST,THENSY,WHILESY,WITHSY]+FSYS);
        ACCEPT(IFSY);
        EXPRESSION([THENSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,THENSY,WHILESY,WITHSY

        FSYS);;
        ACCEPT(THENSY);
        STMT(ACCFSYS,FSYS);
             4:
             if SYM = ELSESY then begin ACCEPT (ELSESY) ; if SYM =
             IFSY then IFSTMT(ACCFSYS,FSYS) else STMT(ACCFSYS,FSYS); goto 4  end ;
        end;
```

fig. 6.5 modified version of IFSTMT .

```
procedure ILLUS;
      begin
           HANDILLUS;
           if CHKSYMSET([ THISSYM , THATSYM ]) then
             begin
                 HAND1
                 (*ND1 OTHER STATEMENTS *)ND1
             end;
           ACCEPT( ARBSYM );
           HAND2;
           while CHKSYMSET([ THISSYM ,THATSYM ]) do
             begin
                 HAND3;
                 (* OTHER STATEMENTS *)
             end;
           ACCEPT( LASTSYM );
           HAND4
      end;
```

prog 7.1 A non-terminal procedure with HAND procedures inserted.

```
procedure ILLUS;
      begin
          HANDILLUS;
          if CHKSYMSET([ THISSYM , THATSYM ]) then
            begin
                HAND1
                (*ND1 OTHER STATEMENTS *)ND1
            end;
          ACCEPT( ARBSYM );
          HAND2;
          while CHKSYMSET([ THISSYM ,THATSYM ]) do
            begin
                HAND3;
                (* OTHER STATEMENTS *)
            end;
          ACCEPT( LASTSYM );
          HAND4
      end;
```

prog 7.1 A Non-terminal procedure with HAND procedures inserted.

APPENDIX A

```
LITERAL
ALL             =   ' '..'~';
LETTER          =   'A'..'Z';
LOWCASE         =   'a'..'z';
EXP             =   'E';
DIGIT           =   '0'..'9';
STOP            =   '.';
QUOTE           =   '''';
EOS             =   '=';
LTS             =   '<';
GTS             =   '>';
NES             =   '#';
PLUSMINUS       =   '+-';
COL             =   ':';
ADDOPM          =   '&';
MUL             =   '*/@';
OTHERS          =   '()[],;^';
BLANK           =   ' ';

TOKEN
IDENT           =   (LETTER|LOWCASE|EXP)(LETTER|LOWCASE|EXP|DIGIT) * ;
INTCONST        =   DIGIT + ;
REALCONST       =   DIGIT + ( STOP DIGIT + | (  STOP DIGIT + | NULL ) EXP
                    ( PLUSMINUS | NULL ) DIGIT + )  ;
ASSIGN          =   COL EOS;
RELOPMEO        =   LTS|GTS|LTS GTS|LTS EOS|GTS EOS|NES;
COLON           =   COL;
PERIOD          =   STOP;
TWODOT          =   STOP STOP;
SIGN            =   PLUSMINUS;
ADDOPMS         =   ADDOPM;
MULOP           =   MUL;
EQ              =   EOS;
STRGCONST       =   QUOTE ( ALL|LETTER|LOWCASE|EXP|DIGIT|STOP|QUOTE QUOTE|
                    EOS|LTS|GTS|NES|PLUSMINUS|COL|ADDOPM|MUL|
                    OTHERS|BLANK ) * QUOTE ;
OTHEROPS        =   OTHERS

DELIMITER
BLANK

KEYWORD
TOKEN OTHEROPS
WL1        =    <'(' LPAREN>,<')' RPAREN>,<'[' LBRAC>,<']' RBRAC>,
                <';' SEMICOL>,<',' COMA>,<'^' ARROW>
TOKEN IDENT
WL2        =    <'array' ARRAYSY>,<'begin' BEGINSY>,<'case' CASESY>,
                <'const' CONSTSY>,<'do' DOSY>,<'downto' DOWNTOSY>,
                <'else' ELSESY>,<'end' ENDSY>,<'file' FILESY>,
                <'for' FORSY>,<'function' FUNCSY>,<'goto' GOTOSY>,
                <'if' IFSY>,<'label' LABELSY>,<'not' NOTSY>,
                <'of' OFSY>,<'packed' PACKEDSY>,<'procedure' PROCSY
                >,<'PROGRAM' PROGRAMSY>,<'record' RECORDSY>,
                <'repeat' REPEATSY>,<'set' SETSY>,<'then' THENSY>,
                <'to' TOSY>,<'type' TYPESY>,<'until' UNTILSY>,
                <'var' VARSY>,<'while' WHILESY>,<'with' WITHSY>,
                <'nil' NILSY>,<'in' INSY>,<'div' DIVSY>,<'mod' MODSY>
                ,<'or' ORSY>,<'and' ANDSY> .
```

Lexicon specification of PASCAL

APPENDIX A

```
 [C+F+]
(<PROG,PROGHEADING,IDLIST,BLOCK,LABELDECPT,CONSTDECPT,
 CONSTDEF,CONSTANT,NUMBER,TYPEDECPT,TYPEDEF,TYPEDENOTER,
 SIMPLETYPE,ENUMTYPE,IDTYPE,SUBTYPE,IDLESSCONST,STRUCTTYPE,
 ARRAYTYPE,RECTYPE,FIELDLIST,VARIANTPT,VARIANT,CONSTLIST,SETTYPE,
 FILETYPE,PTRTYPE,VARDECPT,VARDEF,PROCFNDECPT,PROCDEC,FNDEC,
 PROCHEADING,FNHEADING,FORMPARLIST,FORMPARSPEC,VALVARPARSP,
 STMTPT,COMPSTMT,STMTSEQ,STMT,ASSPRUSTMT,GOTOSTMT,STRUCTSTMT,
 IFSTMT,CASESTMT,CASEBODY,REPSTMT,WHILESTMT,REPEATSTMT,FORSTMT,
 WITHSTMT,RECVARLIST,EXPRESSION,SIMPLEEXP,TERM,FACTOR,
 SETCONSTR,MEMBDESGN,EXPLIST,ACTUALPARA,VARACCESS,ACTPARLIST
                                >,
 <ARRAYSY,BEGINSY,CASESY,CONSTSY,RELOPMEQ(D),EQ(D),DOSY,
  DOWNTOSY,ELSESY,ENDSY,FILESY,FORSY,FUNCSY,
  GOTOSY,IFSY,LABELSY,NOTSY,OFSY,PACKEDSY,PROCSY,
  PROGRAMSY,RECORDSY,REPEATSY,SETSY,THENSY,TOSY,TYPESY,
  UNTILSY,VARSY,WHILESY,WITHSY,SIGN(D),SEMICOL(D),ASSIGN(D),
  COLON(D),PERIOD(D),ARROW(D),LPAREN(D),RPAREN(D),LBRAC(D),RBRAC(D),INTCONST,
  REALCONST,STRGCONST,IDENT,NILSY,COMA(D),TWODOT(D),ADDOPMS(D) ,MULOP(D),
  INSY , DIVSY , MODSY , ANDSY , ORSY
               >,
 <  PROG--> PROGHEADING 'SEMICOL' BLOCK 'PERIOD',
    PROGHEADING--> 'PROGRAMSY' 'IDENT' ['LPAREN' IDLIST 'RPAREN'],
    IDLIST--> 'IDENT' {'COMA' 'IDENT'},
    BLOCK--> LABELDECPT CONSTDECPT TYPEDECPT  VARDECPT PROCFNDECPT STMTPT,
    LABELDECPT-->['LABELSY' 'INTCONST' {'COMA' 'INTCONST'} 'SEMICOL'],
    CONSTDECPT-->['CONSTSY' CONSTDEF 'SEMICOL' {CONSTDEF 'SEMICOL'}],
    CONSTDEF--> 'IDENT' 'EQ' CONSTANT,
    CONSTANT--> ['SIGN']( NUMBER / 'IDENT')/'STRGCONST',
    NUMBER--> 'INTCONST'/'REALCONST',
    TYPEDECPT-->['TYPESY' TYPEDEF 'SEMICOL' { TYPEDEF 'SEMICOL'}],
    TYPEDEF--> 'IDENT' 'EQ' TYPEDENOTER,
    TYPEDENOTER--> SIMPLETYPE / STRUCTTYPE /PTRTYPE,
    SIMPLETYPE--> ENUMTYPE / IDTYPE /SUBTYPE,
    ENUMTYPE--> 'LPAREN' IDLIST 'RPAREN' ,
    IDTYPE--> 'IDENT' [ 'TWODOT' CONSTANT],
    SUBTYPE--> IDLESSCONST 'TWODOT' CONSTANT,
    IDLESSCONST-->'SIGN'('IDENT' / NUMBER)/NUMBER/'STRGCONST',
    STRUCTTYPE--> ['PACKEDSY'] (ARRAYTYPE/RECTYPE/SETTYPE/FILETYPE),
    ARRAYTYPE--> 'ARRAYSY' 'LBRAC' SIMPLETYPE {'COMA' SIMPLETYPE} 'RBRAC' 'OFSY' TYPEDENOTER ,
    RECTYPE--> 'RECORDSY' [FIELDLIST] 'ENDSY' ,
    FIELDLIST--> IDLIST 'COLON' TYPEDENOTER ['SEMICOL' FIELDLIST ]/VARIANTPT,
    VARIANTPT--> 'CASESY' 'IDENT' ['COLON' 'IDENT' ] 'OFSY' VARIANT {'SEMICOL' VARIANT },
    VARIANT-->CONSTLIST 'COLON' 'LPAREN' [FIELDLIST] 'RPAREN' ,
    CONSTLIST--> CONSTANT { 'COMA' CONSTANT},
    SETTYPE-->'SETSY' 'OFSY' SIMPLETYPE,
    FILETYPE--> 'FILESY' 'OFSY' SIMPLETYPE,
      PTRTYPE--> 'ARROW' 'IDENT' ,
    VARDECPT--> ['VARSY' VARDEF 'SEMICOL'  {VARDEF 'SEMICOL' }],
    VARDEF--> IDLIST 'COLON' TYPEDENOTER,
    PROCFNDECPT--> {PROCDEC/FNDEC},
    PROCDEC--> PROCHEADING 'SEMICOL' ('IDENT' /BLOCK) 'SEMICOL' ,
    FNDEC--> FNHEADING 'SEMICOL' ('IDENT' /BLOCK) 'SEMICOL',
    PROCHEADING--> 'PROCSY' 'IDENT' [FORMPARLIST],
    FNHEADING--> 'FUNCSY' 'IDENT' [FORMPARLIST] 'COLON' 'IDENT' ,
    FORMPARLIST--> 'LPAREN' FORMPARSPEC {'SEMICOL' FORMPARSPEC} 'RPAREN' ,
FORMPARSPEC--> VALVARPARSP/PROCHEADING/FNHEADING,
VALVARPARSP-->['VARSY'] IDLIST 'COLON' 'IDENT',
VARACCESS-->{ 'PERIOD' 'IDENT'/'ARROW'/'LBRAC' EXPLIST 'RBRAC'  },
STMTPT--> COMPSTMT,
COMPSTMT--> 'BEGINSY' STMTSEQ 'ENDSY',
STMTSEQ--> STMT {'SEMICOL' STMT},
STMT--> ['INTCONST' 'COLON' ] [ASSPROSTMT/GOTOSTMT/STRUCTSTMT],
ASSPROSTMT--> 'IDENT' [ VARACCESS 'ASSIGN' EXPRESSION /ACTPARLIST],
GOTOSTMT--> 'GOTOSY' 'INTCONST' ,
STRUCTSTMT-->COMPSTMT/IFSTMT/CASESTMT/REPSTMT/WITHSTMT,
IFSTMT--> 'IFSY' EXPRESSION 'THENSY'  STMT  ,
CASESTMT--> 'CASESY' EXPRESSION 'OFSY' CASEBODY 'ENDSY',
CASEBODY--> CONSTLIST 'COLON' STMT { 'SEMICOL' CONSTLIST 'COLON' STMT},
REPSTMT--> WHILESTMT/REPEATSTMT/FORSTMT,
WHILESTMT--> 'WHILESY' EXPRESSION 'DOSY' STMT,
REPEATSTMT--> 'REPEATSY' STMTSEQ 'UNTILSY' EXPRESSION ,
FORSTMT--> 'FORSY' 'IDENT' 'ASSIGN' EXPRESSION ('TOSY'/'DOWNTOSY') EXPRESSION 'DOSY' STMT,
WITHSTMT--> 'WITHSY' RECVARLIST 'DOSY' STMT,
RECVARLIST--> 'IDENT' VARACCESS { 'COMA' 'IDENT' VARACCESS },
EXPRESSION--> SIMPLEEXP {('EQ'/'RELOPMEQ'/'INSY') SIMPLEEXP},
SIMPLEEXP--> ['SIGN'] TERM {( 'ADDOPMS'/'SIGN')  TERM},
TERM--> FACTOR { ('DIVSY'/'MODSY'/'MULOP'/'ORSY'/'ANDSY') FACTOR },
FACTOR--> 'IDENT'( VARACCESS/ACTPARLIST)/'LPAREN' EXPRESSION 'RPAREN'/'NOTSY' FACTOR
          /'NILSY'/SETCONSTR/NUMBER/'STRGCONST',
SETCONSTR--> 'LBRAC' [MEMBDESGN {'COMA' MEMBDESGN}]   'RBRAC' ,
MEMBDESGN--> EXPRESSION [ 'TWODOT' EXPRESSION ],
ACTUALPARA--> EXPRESSION [ 'COLON' EXPRESSION ['COLON' EXPRESSION ]],
EXPLIST--> EXPRESSION { 'COMA' EXPRESSION },
ACTPARLIST-->'LPAREN' ACTUALPARA { 'COMA' ACTUALPARA } 'RPAREN'
          >,PROG)
```

Syntax Specification of PASCAL

APPENDIX A

```
%$D+\ %$P-\
PROGRAM LEXPAR(INPUT,OUTPUT);

  const
       BUFFLGTH = 133;
       MAXERRINLIne = 10; ERRTDLGTH = 10;
       ALFALENGTH= 11;
       LNTH=133;
       LEXSTNUM=    19;       LITMAX=    17;
       TKNMAX= 14;
       NURW= 42;
       NUMTKNWITHW1=   2;
       WLNUM= 10;

  type
       ALPHA=packed array [1..ALFALENGTH] of char;


       SY = (  ILLEGAL  ,    ANDSY          ,   ORSY          ,   MODSY          ,   DIVSY         ,   INSY          ,   NILSY
             ,   WITHSY       ,   WHILESY      ,   VARSY        ,   UNTILSY       ,   TYPESY       ,   TOSY         ,   THENSY
             ,   SETSY        ,   REPEATSY     ,   RECORDSY     ,   PROGRAMSY     ,   PROCSY       ,   PACKEDSY     ,   OFSY
             ,   NOTSY        ,   LABELSY      ,   IFSY         ,   GOTOSY        ,   FUNCSY       ,   FORSY        ,   FILESY
             ,   ENDSY        ,   ELSESY       ,   DOWNTOSY     ,   DOSY          ,   CONSTSY      ,   CASESY       ,   BEGINSY
             ,   ARRAYSY      ,   ARROW        ,   COMA         ,   SEMICOL       ,   RBRAC        ,   LBRAC
             ,   RPAREN       ,   LPAREN       ,   OTHEROPS     ,   STRGCONST     ,   EQ           ,   MULOP
             ,   ADDOPMS      ,   SIGN         ,   TWODOT       ,   PERIOD        ,   COLON        ,   RELOPMEQ     ,   ASSIGN
             ,   REALCONST    ,   INTCONST     ,   IDENT

             ,   EOS   ) ;
       SETOFSYS = set of SY;
       ERRIDBUFF = packed array [1..ERRTDLGTH] of char;
       ERRTYPE = ( INSERTION,REPLACEMENt,LEXICAL,GLOBAL,SINGLEGLOBal,SKIPGLOBAL);
       ERRELMTTYPe=record
                          ERRMSGINDEx:0..MAXERRINLIne;
                          case   ERRCLASS: ERRTYPE of
                               INSERTION:(INSERTSYM:SY);
                               REPLACEMENt:(REPSYM1,REPSYM2:SY);
                               LEXICAL:(ERRNO:integer);
                               GLOBAL:(ERRSYMSET:SETOFSYS;NTNAME:ERRIDBUFF);
                               SINGLEGLOBal:(ERRSYM:SY)
                    end;
       ERRRECDTYPe=array[0..MAXERRINLIne] of ERRELMTTYPe;
       SYMBOL=0..    57;
       TOKENS=0..TKNMAX;
       LITERALS=-1..LITMAX;
       NUMSTLEX=0..LEXSTNUM;
       A=1..NUMTKNWITHW1;
       CURWLTYPE=record
                        TOK:TOKENS; START,SIZE:integer
                 end;
       RESWDS=1..NURW;
       KWSTYPE=record
                      STRING:ALPHA; LEXVAL:SYMBOL
                end;

  var
```

```
    J,K:integer;
    SKIP:integer;
    LIT:array[char] of LITERALS;
    ID:ALPHA;
    CURWL:array[A] of CURWLTYPE;
    HASWL:array[TOKENS] of boolean;
    KEYWORDSPAce:array[RESWDS] of KWSTYPE;
    DELIMSET1:array[-1.. 17] of boolean;
    SYPOS:array[0.. 57] of SY ;
    PREVSYM,NEXTSYM:SY;
    CH:char;
    LEXBGN,LEXSIZE:integer;
    NJ:integer;TYPEOFTERM:integer;
    ERRBUFFER:array[0..1] of array[0..BUFFLGTH] of char;
    BUFFER:array[0..1]of array [0..BUFFLGTH] of char;
    IVALU,TDOLDP,SLGTH,BUFFINDEX,NEXTBUFFINdex:integer;
    RVALU:real;
    SYM:SY;IDNAME:packed array [1..11] of char ;
    LLVALUE:array[0..1] of 0..BUFFLGTH;
    IDLGTH:0..11;
    CC,LL,NEXTLL:0..BUFFLGTH;
    PREVPOSITIon:array[0..1] of 0..BUFFLGTH;
    I,ERRLP,LP,RP,PREVLP,PREVRP,NEXTLP,NEXTRP:integer;
    FIRST:boolean;
    RECOVERY:(LOCAL,NONLOCAL,NOPREVATTMpt);
    ATTMPTRECV,LASTERRGIVen,BLANKLINE,LASTLINE:boolean;
    PROCERRCURsor:0..2;
    ERRRECD:array[0..1] of ERRRECDTYPe;
    LINENO:integer;
    PREVSET:array [SY] of SETOFSYS;
    SYMNAME:array[0..71] of ERRIDBUFF;
    SYMLGTH:array[0..71] of integer;
    ERRVALU:1..5;
    LINE:array[1..133] of char;
    NOOFWARNINgs, NOOFERRS:integer;
    ERRPRESENT:array[0..1] of boolean;
    ERRINLINE:array[0..1] of 0..MAXERRINLine;
function ELMT(S:SETOFSYS):SY;
    extern;
function CARD(S:SETOFSYS):integer;
    extern;
procedure ERROR(ERRELMT:ERRELMTTYPe);
      var
          POSITION:0..BUFFLGTH;XERRINLINE:0..MAXERRINLIne;
      begin
          with ERRELMT do
            begin
                if ERRCLASS#SKIPGLOBAL then
                  begin
                      if ERRCLASS=LEXICAL then POSITION:=CC
                      else POSITION:=LP;
                      ERRBUFFER[BUFFINDEX][POSITION]:='^';
                      XERRINLINE:=ERRINLINE[BUFFINDEX];
                      if XERRINLINE<MAXERRINLIne then
                        begin XERRINLINE:=XERRINLINE+1;
                            if XERRINLINE=MAXERRINLIne then
```

```
                               begin ERRRECD[BUFFINDEX][XERRINLINE].ERRCLASS:=LEXICAL;
                                   ERRRECD[BUFFINDEX][XERRINLINE].ERRNO:=26
                               end
                             else
                               begin
                                   if XERRINLINE=1 then ERRMSGINDEX:=0
                                   else ERRMSGINDEx:=
                                         ERRRECD[BUFFINDEX][XERRINLINE-1].ERRMSGINDEx;
                                   ERRMSGINDEX:=ERRMSGINDEx+1;
                                   ERRRECD[BUFFINDEX][XERRINLINE]:=ERRELMT
                               end;
                             PREVPOSITIon[BUFFINDEX]:=POSITION;
                         end;
                       ERRINLINE[BUFFINDEX]:=XERRINLINE;
                   end;
                 if ((ERRCLASS=LEXICAL) and (ERRNO#25)) or ( ERRCLASS in [INSERTION,REPLACEMENt]) then
                       NOOFWARNINGs:=NOOFWARNINGs+1
                 else NOOFERRS:=NOOFERRS+1
             end;
           ERRPRESENT[BUFFINDEX]:=true
      end;
procedure ERRORMESSAge(ERRELMT:ERRELMTTYPe);
      var
           S:SETOFSYS;E:SY;
      begin
           with ERRELMT do
             begin
                 case ERRCLASS of
                       INSERTION:WRITELN(TTY,SYMNAME[ORD(INSERTSYM)],' TO BE  INSERTED');
                       REPLACEMENt:WRITELN(TTY,SYMNAME[ORD(REPSYM1)],'TO BE REPLACED BY   ',SYMNAME[ORD( REPSYM2)]);
                       SINGLEGLOBal:WRITELN(TTY,SYMNAME[ORD(ERRSYM)],' EXPECTED ');
                       GLOBAL:
                               begin
                                   S:=ERRELMT.ERRSYMSET;E:=ELMT(S);S:=S-[E];WRITE(TTY,SYMNAME[ORD(E)]);
                                   while S#[] do
                                     begin E:=ELMT(S);S:=S-[E];WRITE(TTY,'/',SYMNAME[ORD(E)]);
                                     end;
                                   WRITELN(TTY,' EXPECTED IN ',ERRELMT.NTNAME);
                               end;
                       LEXICAL:
                               begin
                                   case ERRNO of
                                         25:WRITELN(TTY,' PARSER RESTARTED');
                                         26:WRITELN(TTY,'MORE THAN TEN ERRORS IN A LINE');
                                         31:WRITELN(TTY,' DIGIT REQD IN EXPONENT PART');
                                         32:WRITELN(TTY,' DIGIT REQD AFTER DECIMAL');
                                         33:WRITELN(TTY,'RIGHT QUOTE NOT ENCOUNTERED');
                                         34:WRITELN(TTY,'ILLEGAL CHARACTER ENCOUNTERED');
                                         35:WRITELN(TTY,'EOF ENCOUNTERED')
                                   end;
                               end
                 end;
             end;
      end;
procedure PROCESSERRor(BUFFINDEX:integer);
      var
```

```
          I:integer;
          XERRINLINE:integer;
      begin
          if PROCERRCURsor<2 then PROCERRCURsor:=PROCERRCURsor+1
          else
                if ERRPRESENT[BUFFINDEX]   then
                  begin WRITELN(TTY,'    ',BUFFER[BUFFINDEX]:LLVALUE[BUFFINDEX]);
                      WRITELN(TTY,'-->',ERRBUFFER[BUFFINDEX]:LLVALUE[BUFFINDEX]);
                      for I:=1 to LLVALUE[BUFFINDEX]  do ERRBUFFER[BUFFINDEX][I]:=' ';
                      XERRINLINE:=ERRINLINE[BUFFINDEX];
                      if XERRINLINE>0 then
                            for I:=1 to XERRINLINE do
                                  with ERRRECD[BUFFINDEX][I] do
                                    begin WRITE(TTY,' ',ERRMSGINDEx:1,'.^');
                                        ERRORMESSAge(ERRRECD[BUFFINDEX][I]);
                                    end;
                      ERRINLINE[BUFFINDEX]:=0;ERRPRESENT[BUFFINDEX]:=false;
                      PREVPOSITIon[BUFFINDEX]:=0
                  end;
      end;
procedure LEXERROR(N:integer);
      var
          X:ERRELMTTYPe;
      begin     X.ERRCLASS:=LEXICAL;X.ERRNO:=N;ERROR(X)
      end;
procedure ERRORSET(S:SETOFSYS;A:ERRIDBUFF);
      var
          X:ERRELMTTYPe;
      begin  X.ERRCLASS:=GLOBAL;
          X.ERRSYMSET:=S;
          X.NTNAME:=A;
          ERROR(X)
      end;
procedure ERRORSYM(E:SY);
      var
          X:ERRELMTTYPe;
      begin  X.ERRCLASS:=SINGLEGLOBal;
          X.ERRSYM:=E;
          ERROR(X)
      end;
procedure SKIPERROR;
      var
          X:ERRELMTTYPe;
      begin  X.ERRCLASS:=SKIPGLOBAL;
          ERROR(X)
      end;
procedure LOCALERROR(E:SY;ERRCLASSTYpe:ERRTYPE);
      var
          X:ERRELMTTYPe;
      begin  X.ERRCLASS:=ERRCLASSTYpe;
          if X.ERRCLASS=REPLACEMENt then
            begin X.REPSYM1:=E;X.REPSYM2:=PREVSYM
            end
          else X.INSERTSYM:=E;
          ERROR(X)
      end;
```

```
procedure  LEXAN( var SYMNAM:SY);
    forward;
procedure LEXANALYSE;

      procedure RESTORENEXtsym;
            begin  SYM:=NEXTSYM;LP:=NEXTLP;RP:=NEXTRP;BUFFINDEX:=NEXTBUFFINdex;LL:=NEXTLL;
            end;
      begin
          if ATIMPTRECV then
            begin
                RESTORENEXtsym;ATIMPTRECV:=false
            end
          else
            begin
                LP:=CC;
                LEXAN(SYM);
                RP:=CC+1;
            end;
          if RECOVERY = LOCAL then RECOVERY:= NONLOCAL
          else RECOVERY:=NOPREVATIMpt;
          if SYM  = ILLEGAL then
            begin  LEXERROR(34);LEXANALYSE
            end;
          BLANKLINE:=false
      end;
procedure TESTSYS(ACCSYS,STOPSYS:SETOFSYS);
      var
          PREVLP,PREVRP,PREVBUFFINdex,PREVLL:integer;
          S:SETOFSYS;
          TOTSYS:SETOFSYS;
      procedure PRESERVESYm;
            begin PREVSYM:=SYM;PREVLP:=LP;PREVRP:=RP;PREVBUFFINdex:=BUFFINDEX;PREVLL:=LL
            end;
      procedure RESTORESYM;
            begin SYM:=PREVSYM;LP:=PREVLP;RP:=PREVRP;BUFFINDEX:=PREVBUFFINdex;LL:=PREVLL
            end;
      procedure PRESERVENExtsym;
            begin NEXTSYM:=SYM;NEXTLP:=LP;NEXTRP:=RP;NEXTBUFFINdex:=BUFFINDEX;NEXTLL:=LL
            end;
      procedure SKIPSYS;
            begin
                if not (SYM in TOTSYS) then
                  begin  SKIPERROR;
                      while not (SYM in TOTSYS) do
                        begin
                            if ERRBUFFER[BUFFINDEX][LP]=' ' then ERRBUFFER[BUFFINDEX][I-1]:='*';
                            for I:=LP+1  to RP do ERRBUFFER[BUFFINDEX][I]:='*';
                            LEXANALYSE
                        end;
                  end;
                (*ERROR MESSAGE*)
            end;
      begin   (*TESTSYS*)
          ACCSYS:=ACCSYS+[ELSESY];
          if (not (SYM in ACCSYS))  then
            begin
```

```
                TOTSYS:=ACCSYS+STOPSYS;
                if  (RECOVERY<>NONLOCAL) and (not ATTMPTRECV) then
                  begin
                      S:=ACCSYS*PREVSET[SYM];
                      if CARD(S)<=1 then
                        begin
                            if (CARD(S)=1)  then
                              begin
                                    begin PRESERVENExtsym;ATTMPTRECV:=true;SYM:=ELMT(S);
                                        RECOVERY:=LOCAL;    LOCALERROR(SYM,INSERTION);
                                    end
                              end
                            else
                                  if (not (SYM in (TOTSYS))) then
                                    begin
                                        PRESERVESYm;LEXANALYSE;ATTMPTRECV:=true;
                                        PRESERVENExtsym;RESTORESYM;S:=ACCSYS*PREVSET[NEXTSYM];
                                        if (CARD(S)=1)
                                        then
                                          begin SYM:=ELMT(S);
                                              RECOVERY:=LOCAL;LOCALERROR(SYM,REPLACEMENt);
                                          end
                                        else SKIPSYS;
                                    end;
                        end
                      else SKIPSYS;
                  end
                else SKIPSYS;
            end;
      end;
procedure ACCEPT(ACCSYM:SY);
      begin
          if SYM=ACCSYM then LEXANALYSE
          else ERRORSYM(ACCSYM);
      end;
function CHKSYMSET(S:SETOFSYS):boolean;
      begin CHKSYMSET:=SYM in S
      end;


procedure INITSYMNAMes;
      begin
          SYMNAME[0]:='ILLEGAL  ';
          SYMNAME[  1]:='ANDSY     ';
          SYMNAME[  2]:='ORSY      ';
          SYMNAME[  3]:='MODSY     ';
          SYMNAME[  4]:='DIVSY     ';
          SYMNAME[  5]:='INSY      ';
          SYMNAME[  6]:='NILSY     ';
          SYMNAME[  7]:='WITHSY    ';
          SYMNAME[  8]:='WHILESY   ';
          SYMNAME[  9]:='VARSY     ';
          SYMNAME[ 10]:='UNTILSY   ';
          SYMNAME[ 11]:='TYPESY    ';
          SYMNAME[ 12]:='TOSY      ';
          SYMNAME[ 13]:='THENSY    ';
          SYMNAME[ 14]:='SETSY     ';
          SYMNAME[ 15]:='REPEATSY  ';
          SYMNAME[ 16]:='RECORDSY  ';
          SYMNAME[ 17]:='PROGRAMSY ';
          SYMNAME[ 18]:='PROCSY    ';
          SYMNAME[ 19]:='PACKEDSY  ';
          SYMNAME[ 20]:='OFSY      ';
          SYMNAME[ 21]:='NOTSY     ';
          SYMNAME[ 22]:='LABELSY   ';
          SYMNAME[ 23]:='IFSY      ';
          SYMNAME[ 24]:='GOTOSY    ';
          SYMNAME[ 25]:='FUNCSY    ';
          SYMNAME[ 26]:='FORSY     ';
          SYMNAME[ 27]:='FILESY    ';
          SYMNAME[ 28]:='ENDSY     ';
          SYMNAME[ 29]:='ELSESY    ';
          SYMNAME[ 30]:='DOWNTOSY  ';
          SYMNAME[ 31]:='DOSY      ';
          SYMNAME[ 32]:='CONSTSY   ';
          SYMNAME[ 33]:='CASESY    ';
          SYMNAME[ 34]:='BEGINSY   ';
          SYMNAME[ 35]:='ARRAYSY   ';
          SYMNAME[ 36]:='ARROW     ';
          SYMNAME[ 37]:='COMA      ';
          SYMNAME[ 38]:='SEMICOL   ';
          SYMNAME[ 39]:='RBRAC     ';
          SYMNAME[ 40]:='LBRAC     ';
          SYMNAME[ 41]:='RPAREN    ';
          SYMNAME[ 42]:='LPAREN    ';
          SYMNAME[ 43]:='OTHEROPS  ';
          SYMNAME[ 44]:='STRGCONST ';
          SYMNAME[ 45]:='EQ        ';
          SYMNAME[ 46]:='MULOP     ';
          SYMNAME[ 47]:='ADDDPMS   ';
          SYMNAME[ 48]:='SIGN      ';
          SYMNAME[ 49]:='TWODOT    ';
          SYMNAME[ 50]:='PERIOD    ';
          SYMNAME[ 51]:='COLON     ';
          SYMNAME[ 52]:='RELOPMEQ  ';
          SYMNAME[ 53]:='ASSIGN    ';
          SYMNAME[ 54]:='REALCONST ';
          SYMNAME[ 55]:='INTCONST  ';
          SYMNAME[ 56]:='IDENT     ';
          SYMNAME[ 57]:='EOS       ';
      end;

procedure LEXAN;
      label
          0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 100;
          var
          LSIZE:integer;
          FINALORDVAL:TOKENS;
          LITNUMBER:LITERALS;
          LEX:SYMBOL;

      procedure STORE( I:integer);
      begin
          LEXSIZE    :=LSIZE
          FINALORDVAL:=I
      end;


procedure RTEOF;
      begin
          SYM:=EOS;LP:=LLVALUE[BUFFINDEX];LP:=RP;goto 100
      end;
procedure NXTLIT(var LITNUM:LITERALS);
      var
          CH:char;
      begin
          if CC=LL then
                if EOF(INPUT) then
                  begin
                      LITNUM:= -1;
                      if not (LASTERRGIVen) then
                            PROCESSERRor((BUFFINDEX + 1) mod 2 );
                      LASTERRGIVen:=true;
                      RTEOF
                  end
                else
                  begin
                      if ( not BLANKLINE ) then
                        begin BUFFINDEX:= (BUFFINDEX + 1) mod 2 ;
                            PROCESSERRor(BUFFINDEX);
                        end;
                      BLANKLINE:=true;
                      READ(CH);
                      CC:=1;BUFFER[BUFFINDEX][1]:=CH;
                      LL:=1;LINE[LL]:=CH;LINENO:=LINENO + 1;
                      if ( not FIRST ) then LITNUM:=0
                      else
                        begin
                            while EOLN(INPUT) do
                              begin READLN;WRITELN
                              end;
                            if EOF(INPUT) then LITNUM:= -1
                            else
                              begin
                                  while not(EOLN(INPUT)) and (LL<132) do
                                    begin
                                        LL:=LL+1;READ(CH);
                                        BUFFER[BUFFINDEX][LL]:=CH;
                                        LINE[LL]:=CH
                                    end;
                                  LITNUM:=LIT[LINE[1]]
                              end
                        end;
                      if EOLN(INPUT) then
                        begin
                            LL:=LL+1;
                            BUFFER[BUFFINDEX][LL]:=' ';LINE[LL]:=' ' ;
                        end;
                      READLN;
                      LLVALUE[BUFFINDEX]:=LL
                  end
          else
            begin
                CC:=CC+1;
                LITNUM:=LITLINE[CC]];
            end;
          LSIZE:=LSIZE+1;
          FIRST:=false;
      end;
begin
    FINALURDVAL:=0;
    repeat
          FIRST:=true;
          LSIZE:=0;
          NXTLIT(LITNUMBER);
    until not DELIMSET1[LITNUMBER];
    LEXBGN:=CC ;
    if LITNUMBER=-1 then
      begin LEX:= 57; SYMNAM:=SYPOS[LEX]; LSIZE:=0
      end
    else
      begin
          goto  20;
          1:
          NXTLIT(LITNUMBER);
          20:
          if LITNUMBER=  2  then goto   2 ;
          if LITNUMBER=  3  then goto   2 ;
          if LITNUMBER=  4  then goto   2 ;
          if LITNUMBER=  5  then goto   3 ;
          if LITNUMBER=  6  then goto   6 ;
          if LITNUMBER=  7  then goto   8 ;
          if LITNUMBER=  8  then goto  10 ;
          if LITNUMBER=  9  then goto  11 ;
          if LITNUMBER= 10  then goto  13 ;
          if LITNUMBER= 11  then goto  12 ;
          if LITNUMBER= 12  then goto  14 ;
          if LITNUMBER= 13  then goto  15 ;
          if LITNUMBER= 14  then goto  17 ;
          if LITNUMBER= 15  then goto  18 ;
          if LITNUMBER= 16  then goto  19 ;
          goto 0;

          2:
          STORE(  1);
          NXTLIT(LITNUMBER);
          if LITNUMBER=  2  then goto   2 ;
          if LITNUMBER=  3  then goto   2 ;
          if LITNUMBER=  4  then goto   2 ;
          if LITNUMBER=  5  then goto   2 ;
          goto 0;

          3:
          STORE(  2);
```

```
NXTLIT(LITNUMBER);
if LITNUMBER=  4  then goto   4 ;
if LITNUMBER=  5  then goto   3 ;
if LITNUMBER=  6  then goto   4 ;
goto 0;

4:
NXTLIT(LITNUMBER);
if LITNUMBER=  5  then goto   5 ;
if LITNUMBER= 12  then goto   4 ;
goto 0;

5:
STORE(   3);
NXTLIT(LITNUMBER);
if LITNUMBER=  5  then goto   5 ;
goto 0;

6:
STORE(   7);
NXTLIT(LITNUMBER);
if LITNUMBER=  6  then goto   7 ;
goto 0;

7:
STORE(   8);
NXTLIT(LITNUMBER);
goto 0;

8:
NXTLIT(LITNUMBER);
if LITNUMBER=  1  then goto   8 ;
if LITNUMBER=  2  then goto   8 ;
if LITNUMBER=  3  then goto   8 ;
if LITNUMBER=  4  then goto   8 ;
if LITNUMBER=  5  then goto   8 ;
if LITNUMBER=  6  then goto   8 ;
if LITNUMBER=  7  then goto   9 ;
if LITNUMBER=  8  then goto   8 ;
if LITNUMBER=  9  then goto   8 ;
if LITNUMBER= 10  then goto   8 ;
if LITNUMBER= 11  then goto   8 ;
if LITNUMBER= 12  then goto   8 ;
if LITNUMBER= 13  then goto   8 ;
if LITNUMBER= 14  then goto   8 ;
if LITNUMBER= 15  then goto   8 ;
if LITNUMBER= 16  then goto   8 ;
if LITNUMBER= 17  then goto   8 ;
goto 0;

9:
STORE( 13);
NXTLIT(LITNUMBER);
if LITNUMBER=  7  then goto   8 ;
goto 0;
```

```
10:
STORE( 12);
NXTLIT(LITNUMBER);
goto 0;

11:
STORE(  5);
NXTLIT(LITNUMBER);
if LITNUMBER=  8  then goto  12 ;
if LITNUMBER= 10  then goto  12 ;
goto 0;

12:
STORE(  5);
NXTLIT(LITNUMBER);
goto 0;

13:
STORE(  5);
NXTLIT(LITNUMBER);
if LITNUMBER=  8  then goto  12 ;
goto 0;

14:
STORE(  9);
NXTLIT(LITNUMBER);
goto 0;

15:
STORE(  6);
NXTLIT(LITNUMBER);
if LITNUMBER=  8  then goto  16 ;
goto 0;

16:
STORE(  4);
NXTLIT(LITNUMBER);
goto 0;

17:
STORE( 10);
NXTLIT(LITNUMBER);
goto 0;

18:
STORE( 11);
NXTLIT(LITNUMBER);
goto 0;

19:
STORE( 14);
NXTLIT(LITNUMBER);
goto 0;

0:
LEX:=FINALORDVAL;
```

```
                SYMNAM:=SYPOS[LEX];
                if FINALORDVAL#0 then
                  begin     CC:=LEXBGN+LEXSIZE-1;
                      for I:=1 to ALFALENGTH do ID[I]:=' ';
                      for I:= LEXBGN to CC do
                        begin
                            J:=I-LEXBGN+1;
                            if J<=ALFALENGTH then ID[J]:=LINE[I]
                        end;
                      if HASWL[FINALORDVAL] then
                        begin
                            I:=1; while CURWL[I].TOK#LEX do I:=I+1;
                            if CURWL[I].SIZE # 0 then
                              begin
                                  with CURWL[I] do
                                    begin
                                        J:=START; K:=START+SIZE-1
                                    end;
                                  repeat I:=(J+K)div 2;
                                        with KEYWORDSPACe[I] do
                                          begin
                                              if ID<=STRING then K:=I-1;
                                              if ID>=STRING then J:=I+1
                                          end
                                  until J>K;
                                  if J-1>K then LEX:=KEYWORDSPACe[I].LEXVAL
                              end
                        end;
                      SYMNAM:=SYPOS[LEX]
                  end
                else    LEXSIZE:=LSIZE  ;
            end ;
          100:
      end;
    (*PROCEDURE NXTSYM*)


procedure INITSYPOS;
      begin
          SYPOS[57]:=EOS;
          SYPOS[56]:=ANDSY       ;
          SYPOS[55]:=ORSY        ;
          SYPOS[54]:=MODSY       ;
          SYPOS[53]:=DIVSY       ;
          SYPOS[52]:=INSY        ;
          SYPOS[51]:=NILSY       ;
          SYPOS[50]:=WITHSY      ;
          SYPOS[49]:=WHILESY     ;
          SYPOS[48]:=VARSY       ;
          SYPOS[47]:=UNTILSY     ;
          SYPOS[46]:=TYPESY      ;
          SYPOS[45]:=TOSY        ;
          SYPOS[44]:=THENSY      ;
          SYPOS[43]:=SETSY       ;
          SYPOS[42]:=REPEATSY    ;
          SYPOS[41]:=RECORDSY    ;
```

```
               SYPOS[40]:=PROGRAMSY   ;
               SYPOS[39]:=PROCSY      ;
               SYPOS[38]:=PACKEDSY    ;
               SYPOS[37]:=OFSY        ;
               SYPOS[36]:=NOTSY       ;
               SYPOS[35]:=LABELSY     ;
               SYPOS[34]:=IFSY        ;
               SYPOS[33]:=GOTOSY      ;
               SYPOS[32]:=FUNCSY      ;
               SYPOS[31]:=FORSY       ;
               SYPOS[30]:=FILESY      ;
               SYPOS[29]:=ENDSY       ;
               SYPOS[28]:=ELSESY      ;
               SYPOS[27]:=DOWNTOSY    ;
               SYPOS[26]:=DOSY        ;
               SYPOS[25]:=CONSTSY     ;
               SYPOS[24]:=CASESY      ;
               SYPOS[23]:=BEGINSY     ;
               SYPOS[22]:=ARRAYSY     ;
               SYPOS[21]:=ARROW       ;
               SYPOS[20]:=COMA        ;
               SYPOS[19]:=SEMICOL     ;
               SYPOS[18]:=RBRAC       ;
               SYPOS[17]:=LBRAC       ;
               SYPOS[16]:=RPAREN      ;
               SYPOS[15]:=LPAREN      ;
               SYPOS[14]:=OTHEROPS    ;
               SYPOS[13]:=STRGCONST   ;
               SYPOS[12]:=EQ          ;
               SYPOS[11]:=MULOP       ;
               SYPOS[10]:=ADDOPMS     ;
               SYPOS[ 9]:=SIGN        ;
               SYPOS[ 8]:=TWODOT      ;
               SYPOS[ 7]:=PERIOD      ;
               SYPOS[ 6]:=COLON       ;
               SYPOS[ 5]:=RELOPMEQ    ;
               SYPOS[ 4]:=ASSIGN      ;
               SYPOS[ 3]:=REALCONST   ;
               SYPOS[ 2]:=INTCONST    ;
               SYPOS[ 1]:=IDENT       ;
               SYPOS[0]:=ILLEGAL;
          end;
procedure MAKEREADY;
          begin
               with CURWL[  1] do
                 begin
                     TOK:= 14; START:=  1; SIZE:=  7
                 end;

               with CURWL[  2] do
                 begin
                     TOK:=  1; START:=  8; SIZE:= 35
                 end;

          end;
```

```
procedure INITIALISE;
      begin
          MAKEREADY;
          LIT[' ']:= 17;    LIT['!']:=  1;    LIT['"']:=  1;    LIT['#']:= 11;    LIT['$']:=  1;
          LIT['%']:=  1;    LIT['&']:= 14;    LIT['''']:=  7;    LIT['(']:= 16;    LIT[')']:= 16;
          LIT['*']:= 15;    LIT['+']:= 12;    LIT[',']:= 16;    LIT['-']:= 12;    LIT['.']:=  6;
          LIT['/']:= 15;    LIT['0']:=  5;    LIT['1']:=  5;    LIT['2']:=  5;    LIT['3']:=  5;
          LIT['4']:=  5;    LIT['5']:=  5;    LIT['6']:=  5;    LIT['7']:=  5;    LIT['8']:=  5;
          LIT['9']:=  5;    LIT[':']:= 13;    LIT[';']:= 16;    LIT['<']:=  9;    LIT['=']:=  8;
          LIT['>']:= 10;    LIT['?']:=  1;    LIT['@']:= 15;    LIT['A']:=  2;    LIT['B']:=  2;
          LIT['C']:=  2;    LIT['D']:=  2;    LIT['E']:=  4;    LIT['F']:=  2;    LIT['G']:=  2;
          LIT['H']:=  2;    LIT['I']:=  2;    LIT['J']:=  2;    LIT['K']:=  2;    LIT['L']:=  2;
          LIT['M']:=  2;    LIT['N']:=  2;    LIT['O']:=  2;    LIT['P']:=  2;    LIT['Q']:=  2;
          LIT['R']:=  2;    LIT['S']:=  2;    LIT['T']:=  2;    LIT['U']:=  2;    LIT['V']:=  2;
          LIT['W']:=  2;    LIT['X']:=  2;    LIT['Y']:=  2;    LIT['Z']:=  2;    LIT['[']:= 16;
          LIT['\']:=  1;    LIT[']']:= 16;    LIT['^']:= 16;    LIT['_']:=  1;    LIT['`']:=  1;
          LIT['a']:=  3;    LIT['b']:=  3;    LIT['c']:=  3;    LIT['d']:=  3;    LIT['e']:=  3;
          LIT['f']:=  3;    LIT['g']:=  3;    LIT['h']:=  3;    LIT['i']:=  3;    LIT['j']:=  3;
          LIT['k']:=  3;    LIT['l']:=  3;    LIT['m']:=  3;    LIT['n']:=  3;    LIT['o']:=  3;
          LIT['p']:=  3;    LIT['q']:=  3;    LIT['r']:=  3;    LIT['s']:=  3;    LIT['t']:=  3;
          LIT['u']:=  3;    LIT['v']:=  3;    LIT['w']:=  3;    LIT['x']:=  3;    LIT['y']:=  3;
          LIT['z']:=  3;    LIT['{']:=  1;    LIT['|']:=  1;    LIT['}']:=  1;    LIT['~']:=  1;

          with KEYWORDSPACE[  1] do
            begin
              STRING:='[               '; LEXVAL:= 17
            end;

          with KEYWORDSPACE[  2] do
            begin
              STRING:=']               '; LEXVAL:= 18
            end;

          with KEYWORDSPACE[  3] do
            begin
              STRING:='^               '; LEXVAL:= 21
            end;

          with KEYWORDSPACE[  4] do
            begin
              STRING:='(               '; LEXVAL:= 15
            end;

          with KEYWORDSPACE[  5] do
            begin
              STRING:=')               '; LEXVAL:= 16
            end;

          with KEYWORDSPACE[  6] do
            begin
              STRING:=',               '; LEXVAL:= 20
            end;

          with KEYWORDSPACE[  7] do
            begin
      STRING:=';          '; LEXVAL:= 19
  end;

with KEYWORDSPACE[  8] do
  begin
      STRING:='PROGRAM    '; LEXVAL:= 40
  end;

with KEYWORDSPACE[  9] do
  begin
      STRING:='and        '; LEXVAL:= 56
  end;

with KEYWORDSPACE[ 10] do
  begin
      STRING:='array      '; LEXVAL:= 22
  end;

with KEYWORDSPACE[ 11] do
  begin
      STRING:='begin      '; LEXVAL:= 23
  end;

with KEYWORDSPACE[ 12] do
  begin
      STRING:='case       '; LEXVAL:= 24
  end;

with KEYWORDSPACE[ 13] do
  begin
      STRING:='const      '; LEXVAL:= 25
  end;

with KEYWORDSPACE[ 14] do
  begin
      STRING:='div        '; LEXVAL:= 53
  end;

with KEYWORDSPACE[ 15] do
  begin
      STRING:='do         '; LEXVAL:= 26
  end;

with KEYWORDSPACE[ 16] do
  begin
      STRING:='downto     '; LEXVAL:= 27
  end;

with KEYWORDSPACE[ 17] do
  begin
      STRING:='else       '; LEXVAL:= 28
  end;

with KEYWORDSPACE[ 18] do
  begin
      STRING:='end        '; LEXVAL:= 29
  end;

with KEYWORDSPACE[ 19] do
  begin
      STRING:='file       '; LEXVAL:= 30
  end;

with KEYWORDSPACE[ 20] do
  begin
      STRING:='for        '; LEXVAL:= 31
  end;

with KEYWORDSPACE[ 21] do
  begin
      STRING:='function   '; LEXVAL:= 32
  end;

with KEYWORDSPACE[ 22] do
  begin
      STRING:='goto       '; LEXVAL:= 33
  end;

with KEYWORDSPACE[ 23] do
  begin
      STRING:='if         '; LEXVAL:= 34
  end;

with KEYWORDSPACE[ 24] do
  begin
      STRING:='in         '; LEXVAL:= 52
  end;

with KEYWORDSPACE[ 25] do
  begin
      STRING:='label      '; LEXVAL:= 35
  end;

with KEYWORDSPACE[ 26] do
  begin
      STRING:='mod        '; LEXVAL:= 54
  end;

with KEYWORDSPACE[ 27] do
  begin
      STRING:='nil        '; LEXVAL:= 51
  end;

with KEYWORDSPACE[ 28] do
  begin
      STRING:='not        '; LEXVAL:= 36
  end;

with KEYWORDSPACE[ 29] do
  begin
      STRING:='of         '; LEXVAL:= 37
  end;
```

```
with KEYWORDSPACE[ 30] do
  begin
      STRING:='or        '; LEXVAL:= 55
  end;

with KEYWORDSPACE[ 31] do
  begin
      STRING:='packed    '; LEXVAL:= 38
  end;

with KEYWORDSPACE[ 32] do
  begin
      STRING:='procedure '; LEXVAL:= 39
  end;

with KEYWORDSPACE[ 33] do
  begin
      STRING:='record    '; LEXVAL:= 41
  end;

with KEYWORDSPACE[ 34] do
  begin
      STRING:='repeat    '; LEXVAL:= 42
  end;

with KEYWORDSPACE[ 35] do
  begin
      STRING:='set       '; LEXVAL:= 43
  end;

with KEYWORDSPACE[ 36] do
  begin
      STRING:='then      '; LEXVAL:= 44
  end;

with KEYWORDSPACE[ 37] do
  begin
      STRING:='to        '; LEXVAL:= 45
  end;

with KEYWORDSPACE[ 38] do
  begin
      STRING:='type      '; LEXVAL:= 46
  end;

with KEYWORDSPACE[ 39] do
  begin
      STRING:='until     '; LEXVAL:= 47
  end;

with KEYWORDSPACE[ 40] do
  begin
      STRING:='var       '; LEXVAL:= 48
  end;
```

```
          with KEYWORDSPAce[ 41] do
            begin
               STRING:='while      '; LEXVAL:= 49
            end;

          with KEYWORDSPAce[ 42] do
            begin
               STRING:='with       '; LEXVAL:= 50
            end;

          for I:=0 to TKNMAX do HASWL[I]:=false;
          HASWL[  1]:=true;
          HASWL[ 14]:=true;


          for I:= -1 to 17 do DELIMSET1[I]:=false;
          DELIMSET1[ 17]:=true;
       end;

procedure INITPREVSEts;
       begin
          PREVSET[ADDOPMS   ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[ANDSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[ARRAYSY]:=[COLON    ,EQ    ,OFSY,PACKEDSY] ;
          PREVSET[ARROW    ]:=[ARROW    ,COLON    ,EQ    ,IDENT,OFSY,RBRAC    ] ;
          PREVSET[ASSIGN   ]:=[ARROW    ,IDENT,RBRAC    ] ;
          PREVSET[BEGINSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[CASESY]:=[BEGINSY,COLON    ,DOSY,LPAREN    ,RECORDSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[COLON    ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[COMA     ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[CONSTSY]:=[SEMICOL    ] ;
          PREVSET[DIVSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[DOSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[DOWNTOSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[ELSESY]:=[] ;
          PREVSET[ENDSY]:=[ARROW    ,BEGINSY,COLON    ,DOSY,ENDSY,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RECORDSY,
          RPAREN    ,SEMICOL    ,STRGCONST,THENSY] ;
          PREVSET[EQ    ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[FILESY]:=[COLON    ,EQ    ,OFSY,PACKEDSY] ;
          PREVSET[FORSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[FUNCSY]:=[LPAREN    ,SEMICOL    ] ;
          PREVSET[GOTOSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[IDENT]:=[ADDOPMS    ,ANDSY,ARROW    ,ASSIGN    ,BEGINSY,CASESY,COLON    ,COMA    ,CONSTSY,DIVSY,DOSY,
          DOWNTOSY,EQ    ,FORSY,FUNCSY,IFSY,INSY,LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,OFSY,ORSY,PERIOD    ,PROCSY

          PROGRAMSY,RECORDSY,RELOPMEQ    ,REPEATSY,SEMICOL    ,SIGN    ,THENSY,TOSY,TWODOT    ,TYPESY,UNTILSY,VARSY,
          WHILESY,WITHSY] ;
          PREVSET[IFSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[INSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[INTCONST]:=[ADDOPMS    ,ANDSY,ASSIGN    ,BEGINSY,CASESY,COLON    ,COMA    ,DIVSY,DOSY,DOWNTOSY,EQ

          GOTOSY,IFSY,INSY,LABELSY,LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,OFSY,ORSY,RELOPMEQ    ,REPEATSY,SEMICOL
          ,SIGN    ,THENSY,TOSY,TWODOT    ,UNTILSY,WHILESY] ;
          PREVSET[LABELSY]:=[SEMICOL    ] ;
          PREVSET[LBRAC    ]:=[ADDOPMS    ,ANDSY,ARRAYSY,ARROW    ,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,
          EQ    ,IDENT,IFSY,INSY,LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,ORSY,RBRAC    ,RELOPMEQ    ,SIGN    ,TOSY,
          TWODOT    ,UNTILSY,WHILESY] ;
          PREVSET[LPAREN    ]:=[ADDOPMS    ,ANDSY,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,EQ    ,IDENT,IFSY,
          INSY,LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,OFSY,ORSY,RELOPMEQ    ,SIGN    ,TOSY,TWODOT    ,UNTILSY,
          WHILESY] ;
          PREVSET[MODSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[MULOP    ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[NILSY]:=[ADDOPMS    ,ANDSY,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,EQ    ,IFSY,INSY,LBRAC
          ,LPAREN    ,MODSY,MULOP    ,NOTSY,ORSY,RELOPMEQ    ,SIGN    ,TOSY,TWODOT    ,UNTILSY,WHILESY] ;
          PREVSET[NOTSY]:=[ADDOPMS    ,ANDSY,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,EQ    ,IFSY,INSY,LBRAC
          ,LPAREN    ,MODSY,MULOP    ,NOTSY,ORSY,RELOPMEQ    ,SIGN    ,TOSY,TWODOT    ,UNTILSY,WHILESY] ;
          PREVSET[OFSY]:=[ARROW    ,FILESY,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,SETSY,STRGCONST] ;
          PREVSET[ORSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[PACKEDSY]:=[COLON    ,EQ    ,OFSY] ;
          PREVSET[PERIOD    ]:=[ARROW    ,ENDSY,IDENT,RBRAC    ] ;
          PREVSET[PROCSY]:=[LPAREN    ,SEMICOL    ] ;
          PREVSET[PROGRAMSY]:=[] ;
          PREVSET[RBRAC    ]:=[ARROW    ,IDENT,INTCONST,LBRAC    ,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[REALCONST]:=[ADDOPMS    ,ANDSY,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,EQ    ,IFSY,INSY,
          LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,OFSY,ORSY,RELOPMEQ    ,SEMICOL    ,SIGN    ,TOSY,TWODOT    ,UNTILSY,
          WHILESY] ;
          PREVSET[RECORDSY]:=[COLON    ,EQ    ,OFSY,PACKEDSY] ;
          PREVSET[RELOPMEQ    ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[REPEATSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[RPAREN    ]:=[ARROW    ,ENDSY,IDENT,INTCONST,LPAREN    ,NILSY,RBRAC    ,REALCONST,RPAREN    ,
          STRGCONST] ;
          PREVSET[SEMICOL    ]:=[ARROW    ,BEGINSY,COLON    ,DOSY,ENDSY,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,
          REPEATSY,RPAREN    ,SEMICOL    ,STRGCONST,THENSY] ;
          PREVSET[SETSY]:=[COLON    ,EQ    ,OFSY,PACKEDSY] ;
          PREVSET[SIGN    ]:=[ARROW    ,ASSIGN    ,CASESY,COLON    ,COMA    ,DOWNTOSY,EQ    ,IDENT,IFSY,INSY,INTCONST,
          LBRAC    ,LPAREN    ,NILSY,OFSY,RBRAC    ,REALCONST,RELOPMEQ    ,RPAREN    ,SEMICOL    ,STRGCONST,TOSY,TWODOT
          ,UNTILSY,WHILESY] ;
          PREVSET[STRGCONST]:=[ADDOPMS    ,ANDSY,ASSIGN    ,CASESY,COLON    ,COMA    ,DIVSY,DOWNTOSY,EQ    ,IFSY,INSY,
          LBRAC    ,LPAREN    ,MODSY,MULOP    ,NOTSY,OFSY,ORSY,RELOPMEQ    ,SEMICOL    ,SIGN    ,TOSY,TWODOT    ,UNTILSY,
          WHILESY] ;
          PREVSET[THENSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[TOSY]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[TWODOT    ]:=[ARROW    ,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,RPAREN    ,STRGCONST] ;
          PREVSET[TYPESY]:=[SEMICOL    ] ;
          PREVSET[UNTILSY]:=[ARROW    ,COLON    ,DOSY,ENDSY,IDENT,INTCONST,NILSY,RBRAC    ,REALCONST,REPEATSY,RPAREN
          ,SEMICOL    ,STRGCONST,THENSY] ;
          PREVSET[VARSY]:=[LPAREN    ,SEMICOL    ] ;
          PREVSET[WHILESY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[WITHSY]:=[BEGINSY,COLON    ,DOSY,REPEATSY,SEMICOL    ,THENSY] ;
          PREVSET[EOS]:=[PERIOD    ] ;
      end;
procedure BLOCK(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure PROGHEADING(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure PROG(ACCFSYS,FSYS:SETOFSYS);

      begin
          TESTSYS([PROGRAMSY],[BEGINSY,CONSTSY,FUNCSY,LABELSY,PERIOD    ,PROCSY,SEMICOL    ,TYPESY,VARSY]+FSYS);
          PROGHEADING([SEMICOL    ],[BEGINSY,CONSTSY,FUNCSY,LABELSY,PERIOD    ,PROCSY,SEMICOL    ,TYPESY,VARSY]+FSYS)

          ;
          ACCEPT(SEMICOL    );
          BLOCK([PERIOD    ],[PERIOD    ]+FSYS);;
          ACCEPT(PERIOD    );
          TESTSYS(ACCFSYS,FSYS);
      end;
procedure IDLIST(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure PROGHEADING;

      begin
          TESTSYS([PROGRAMSY],FSYS);
          if CHKSYMSET(  (([PROGRAMSY]) )) then
            begin
                ACCEPT(PROGRAMSY);
                TESTSYS([IDENT],FSYS);;
                ACCEPT(IDENT);
                TESTSYS([LPAREN    ]+ACCFSYS,FSYS);;
                if  CHKSYMSET (([LPAREN    ]) ) then
                  begin
                      ACCEPT(LPAREN    );
                      IDLIST([RPAREN    ],[RPAREN    ]+FSYS);;
                      ACCEPT(RPAREN    );
                      TESTSYS(ACCFSYS,FSYS);
                  end

            end
          else ERRORSET([PROGRAMSY],'PROGHEADIN')
      end;
procedure IDLIST;

      begin
          TESTSYS([IDENT],FSYS);
          if CHKSYMSET(  (([IDENT]) )) then
            begin
                ACCEPT(IDENT);
                TESTSYS([COMA    ]+ACCFSYS,FSYS);;
                while  CHKSYMSET (([COMA    ]) ) do
                  begin
                      ACCEPT(COMA    );
                      TESTSYS([IDENT],FSYS);;
                      ACCEPT(IDENT);
                      TESTSYS([COMA    ]+ACCFSYS,FSYS);
                  end

            end
          else ERRORSET([IDENT],'IDLTST    ')
      end;
procedure CONSTDECPT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure LABELDECPT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure PROCFNDECPt(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STMTPT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure TYPEDECPT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure VARDECPT(ACCFSYS,FSYS:SETOFSYS);
```

```
    forward;
procedure BLOCK;

      begin
          TESTSYS([BEGINSY,CONSTSY,FUNCSY,LABELSY,PROCSY,TYPESY,VARSY],FSYS);
          LABELDECPT([BEGINSY,CONSTSY,FUNCSY,PROCSY,TYPESY,VARSY],[BEGINSY,CONSTSY,FUNCSY,PROCSY,TYPESY,VARSY]+
          FSYS);;
          CONSTDECPT([BEGINSY,FUNCSY,PROCSY,TYPESY,VARSY],[BEGINSY,FUNCSY,PROCSY,TYPESY,VARSY]+FSYS);;
          TYPEDECPT([BEGINSY,FUNCSY,PROCSY,VARSY],[BEGINSY,FUNCSY,PROCSY,VARSY]+FSYS);;
          VARDECPT([BEGINSY,FUNCSY,PROCSY],[BEGINSY,FUNCSY,PROCSY]+FSYS);;
          PROCFUNDECPT([BEGINSY],[BEGINSY]+FSYS);;
          STMTPT(ACCFSYS,FSYS);
      end;
procedure LABELDECPT;

      begin
          TESTSYS([LABELSY]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([LABELSY]) )) then
                  begin
                      if  CHKSYMSET (([LABELSY]) ) then
                        begin
                            if CHKSYMSET(  (([LABELSY]) )) then
                              begin
                                  ACCEPT(LABELSY);
                                  TESTSYS([INTCONST],[SEMICOL   ]+FSYS);;
                                  ACCEPT(INTCONST);
                                  TESTSYS([COMA    ,SEMICOL   ],FSYS);;
                                  while  CHKSYMSET (([COMA    ]) ) do
                                    begin
                                        ACCEPT(COMA    );
                                        TESTSYS([INTCONST],[SEMICOL   ]+FSYS);;
                                        ACCEPT(INTCONST);
                                        TESTSYS([COMA    ,SEMICOL   ],FSYS);
                                    end
                                        ;
                                  ACCEPT(SEMICOL   );
                                  TESTSYS(ACCFSYS,FSYS);
                              end
                        end

                  end
            end
      end;
procedure CONSTDEF(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure CONSTDECPT;

      begin
          TESTSYS([CONSTSY]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([CONSTSY]) )) then
                  begin
```

```
                      if  CHKSYMSET (([CONSTSY]) ) then
                        begin
                            if CHKSYMSET(  (([CONSTSY]) )) then
                              begin
                                  ACCEPT(CONSTSY);
                                  CONSTDEF([SEMICOL   ],[SEMICOL   ]+FSYS);;
                                  ACCEPT(SEMICOL    );
                                  TESTSYS([IDENT]+ACCFSYS,FSYS);;
                                  while  CHKSYMSET (([IDENT]) ) do
                                    begin
                                        CONSTDEF([SEMICOL   ],[SEMICOL   ]+FSYS);;
                                        ACCEPT(SEMICOL    );
                                        TESTSYS([IDENT]+ACCFSYS,FSYS);
                                    end

                              end
                        end

                  end
            end
      end;
procedure CONSTANT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure CONSTDEF;

      begin
          TESTSYS([IDENT],[EQ    ,INTCONST,REALCONST,SIGN    ,STRGCONST]+FSYS);
          ACCEPT(IDENT);
          TESTSYS([EQ    ],[IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST]+FSYS);;
          ACCEPT(EQ    );
          CONSTANT(ACCFSYS,FSYS);
      end;
procedure NUMBER(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure CONSTANT;

      begin
          TESTSYS([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],FSYS);
          if CHKSYMSET(  (([IDENT,INTCONST,REALCONST,SIGN    ]) )) then
            begin
                if  CHKSYMSET (([SIGN    ]) ) then
                  begin
                      ACCEPT(SIGN    );
                      TESTSYS([IDENT,INTCONST,REALCONST],FSYS);
                  end
                      ;
                if CHKSYMSET(  (([INTCONST,REALCONST]) )) then
                  begin
                      NUMBER(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([IDENT]) )) then
                        begin
                            ACCEPT(IDENT);
                            TESTSYS(ACCFSYS,FSYS);
                        end
                      else ERRORSET([IDENT,INTCONST,REALCONST],'CONSTANT  ')
            end
          else
                if CHKSYMSET(  (([STRGCONST]) )) then
                  begin
                      ACCEPT(STRGCONST);
                      TESTSYS(ACCFSYS,FSYS);
                  end
                else ERRORSET([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],'CONSTANT  ')
      end;
procedure NUMBER;

      begin
          TESTSYS([INTCONST,REALCONST],FSYS);
          if CHKSYMSET(  (([INTCONST]) )) then
            begin
                ACCEPT(INTCONST);
                TESTSYS(ACCFSYS,FSYS);
            end
          else
                if CHKSYMSET(  (([REALCONST]) )) then
                  begin
                      ACCEPT(REALCONST);
                      TESTSYS(ACCFSYS,FSYS);
                  end
                else ERRORSET([INTCONST,REALCONST],'NUMBER    ')
      end;
procedure TYPEDEF(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure TYPEDECPT;

      begin
          TESTSYS([TYPESY]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([TYPESY]) )) then
                  begin
                      if  CHKSYMSET (([TYPESY]) ) then
                        begin
                            if CHKSYMSET(  (([TYPESY]) )) then
                              begin
                                  ACCEPT(TYPESY);
                                  TYPEDEF([SEMICOL    ],[SEMICOL    ]+FSYS);;
                                  ACCEPT(SEMICOL    );
                                  TESTSYS([IDENT]+ACCFSYS,FSYS);;
                                  while  CHKSYMSET (([IDENT]) ) do
                                    begin
                                        TYPEDEF([SEMICOL    ],[SEMICOL    ]+FSYS);;
                                        ACCEPT(SEMICOL    );
                                        TESTSYS([IDENT]+ACCFSYS,FSYS);
                                    end

                              end
                        end

                  end
            end
      end;
procedure TYPEDENOTER(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure TYPEDEF;

      begin
          TESTSYS([IDENT],[ARRAYSY,ARROW    ,EQ    ,FILESY,INTCONST,LPAREN    ,PACKEDSY,REALCONST,RECORDSY,SETSY,SIGN
          ,STRGCONST]+FSYS);
          ACCEPT(IDENT);
          TESTSYS([EQ    ],[ARRAYSY,ARROW    ,FILESY,IDENT,INTCONST,LPAREN    ,PACKEDSY,REALCONST,RECORDSY,SETSY,SIGN
          ,STRGCONST]+FSYS);;
          ACCEPT(EQ    );
          TYPEDENOTER(ACCFSYS,FSYS);
      end;
procedure PTRTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure SIMPLETYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STRUCTTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure TYPEDENOTER;

      begin
          TESTSYS([ARRAYSY,ARROW    ,FILESY,IDENT,INTCONST,LPAREN    ,PACKEDSY,REALCONST,RECORDSY,SETSY,SIGN
                   STRGCONST],FSYS);
          if CHKSYMSET(  (([IDENT,INTCONST,LPAREN    ,REALCONST,SIGN    ,STRGCONST]) )) then
            begin
                SIMPLETYPE(ACCFSYS,FSYS);
            end
          else
                if CHKSYMSET(  (([ARRAYSY,FILESY,PACKEDSY,RECORDSY,SETSY]) )) then
                  begin
                      STRUCTTYPE(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([ARROW    ]) )) then
                        begin
                            PTRTYPE(ACCFSYS,FSYS);
                        end
                      else ERRORSET([ARRAYSY,ARROW    ,FILESY,IDENT,INTCONST,LPAREN    ,PACKEDSY,REALCONST,RECORDSY,
                                    SETSY,SIGN    ,STRGCONST],'TYPEDENOTE')
      end;
procedure ENUMTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure IDTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure SUBTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure SIMPLETYPE;

      begin
          TESTSYS([IDENT,INTCONST,LPAREN    ,REALCONST,SIGN    ,STRGCONST],FSYS);
          if CHKSYMSET(  (([LPAREN    ]) )) then
            begin
                ENUMTYPE(ACCFSYS,FSYS);
            end
          else
                if CHKSYMSET(  (([IDENT]) )) then
                  begin
                      IDTYPE(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([INTCONST,REALCONST,SIGN    ,STRGCONST]) )) then
                        begin
                            SUBTYPE(ACCFSYS,FSYS);
                        end
                      else ERRORSET([IDENT,INTCONST,LPAREN    ,REALCONST,SIGN    ,STRGCONST],'SIMPLETYPE')
      end;
procedure ENUMTYPE;

      begin
          TESTSYS([LPAREN    ],[IDENT,RPAREN    ]+FSYS);
          ACCEPT(LPAREN    );
          IDLIST([RPAREN    ],[RPAREN    ]+FSYS);;
          ACCEPT(RPAREN    );
          TESTSYS(ACCFSYS,FSYS);
      end;
procedure IDTYPE;

      begin
          TESTSYS([IDENT],FSYS);
          if CHKSYMSET(  (([IDENT]) )) then
            begin
                ACCEPT(IDENT);
                TESTSYS([TWODOT    ]+ACCFSYS,FSYS);;
                if  CHKSYMSET (([TWODOT    ]) ) then
                  begin
                      ACCEPT(TWODOT    );
                      CONSTANT(ACCFSYS,FSYS);
                  end

            end
          else ERRORSET([IDENT],'IDTYPE     ')
      end;
procedure IDLESSCONSt(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure SUBTYPE;

      begin
          TESTSYS([INTCONST,REALCONST,SIGN    ,STRGCONST],[IDENT,TWODOT    ]+FSYS);
          IDLESSCONSt([TWODOT    ],[IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST,TWODOT    ]+FSYS);;
          ACCEPT(TWODOT    );
          CONSTANT(ACCFSYS,FSYS);
      end;
procedure IDLESSCONSt;

      begin
          TESTSYS([INTCONST,REALCONST,SIGN    ,STRGCONST],FSYS);
          if CHKSYMSET(  (([SIGN    ]) )) then
            begin
                ACCEPT(SIGN    );
                TESTSYS([IDENT,INTCONST,REALCONST],FSYS);
                if CHKSYMSET(  (([IDENT]) )) then
                  begin
                      ACCEPT(IDENT);
                      TESTSYS(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([INTCONST,REALCONST]) )) then
                        begin
                            NUMBER(ACCFSYS,FSYS);
                        end
                      else ERRORSET([IDENT,INTCONST,REALCONST],'IDLESSCONS')
            end
          else
                if CHKSYMSET(  (([INTCONST,REALCONST]) )) then
                  begin
                      NUMBER(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([STRGCONST]) )) then
                        begin
                            ACCEPT(STRGCONST);
                            TESTSYS(ACCFSYS,FSYS);
                        end
                      else ERRORSET([INTCONST,REALCONST,SIGN   ,STRGCONST],'IDLESSCONS')
      end;
procedure ARRAYTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure FILETYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure RECTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure SETTYPE(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STRUCTTYPE;

      begin
          TESTSYS([ARRAYSY,FILESY,PACKEDSY,RECORDSY,SETSY],FSYS);
          if CHKSYMSET(  (([ARRAYSY,FILESY,PACKEDSY,RECORDSY,SETSY]) )) then
            begin
                if  CHKSYMSET (([PACKEDSY]) ) then
                  begin
                      ACCEPT(PACKEDSY)
                  end
                      ;
                if CHKSYMSET(  (([ARRAYSY]) )) then
                  begin
                      ARRAYTYPE(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([RECORDSY]) )) then
                        begin
                            RECTYPE(ACCFSYS,FSYS);
                        end
                      else
                            if CHKSYMSET(  (([SETSY]) )) then
```

```
                              begin
                                  SETTYPE(ACCFSYS,FSYS);
                              end
                            else
                                  if CHKSYMSET(  (([FILESY]) )) then
                                    begin
                                        FILETYPE(ACCFSYS,FSYS);
                                    end
                                  else ERRORSET([ARRAYSY,FILESY,RECORDSY,SETSY],'STRUCTTYPE')
            end
          else ERRORSET([ARRAYSY,FILESY,PACKEDSY,RECORDSY,SETSY],'STRUCTTYPE')
      end;
procedure ARRAYTYPE;

      begin
          TESTSYS([ARRAYSY],FSYS);
          if CHKSYMSET(  (([ARRAYSY]) )) then
            begin
                ACCEPT(ARRAYSY);
                TESTSYS([LBRAC    ],[ARRAYSY,ARROW    ,FILESY,IDENT,INTCONST,LPAREN    ,OFSY,PACKEDSY,RBRAC
                REALCONST,RECORDSY,SETSY,SIGN    ,STRGCONST]+FSYS);;
                ACCEPT(LBRAC    );
                SIMPLETYPE([COMA    ,RBRAC    ],[ARRAYSY,ARROW    ,COMA    ,FILESY,IDENT,INTCONST,LPAREN    ,OFSY,
                PACKEDSY,RBRAC    ,REALCONST,RECORDSY,SETSY,SIGN    ,STRGCONST]+FSYS);;
                while  CHKSYMSET (([COMA    ]) ) do
                  begin
                      ACCEPT(COMA    );
                      SIMPLETYPE([COMA    ,RBRAC    ],[ARRAYSY,ARROW    ,COMA    ,FILESY,IDENT,INTCONST,LPAREN    ,OFSY

                      PACKEDSY,RBRAC    ,REALCONST,RECORDSY,SETSY,SIGN    ,STRGCONST]+FSYS);
                  end
                      ;
                ACCEPT(RBRAC    );
                TESTSYS([OFSY],[ARRAYSY,ARROW    ,FILESY,IDENT,INTCONST,LPAREN    ,PACKEDSY,REALCONST,RECORDSY,SETSY

                SIGN    ,STRGCONST]+FSYS);;
                ACCEPT(OFSY);
                TYPEDENOTER(ACCFSYS,FSYS);
            end
          else ERRORSET([ARRAYSY],'ARRAYTYPE ')
      end;
procedure FIELDLIST(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure RECTYPE;

      begin
          TESTSYS([RECORDSY],FSYS);
          if CHKSYMSET(  (([RECORDSY]) )) then
            begin
                ACCEPT(RECORDSY);
                TESTSYS([CASESY,ENDSY,IDENT],FSYS);;
                if  CHKSYMSET (([CASESY,IDENT]) ) then
                  begin
                      FIELDLIST([ENDSY],[ENDSY]+FSYS);
                  end
                      ;
                ACCEPT(ENDSY);
                TESTSYS(ACCFSYS,FSYS);
            end
          else ERRORSET([RECORDSY],'RECTYPE    ')
      end;
procedure VARIANTPT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure FIELDLIST;

      begin
          TESTSYS([CASESY,IDENT],FSYS);
          if CHKSYMSET(   (([IDENT]) )) then
            begin
                IDLIST([COLON    ],[ARRAYSY,ARROW    ,COLON    ,FILESY,IDENT,INTCONST,LPAREN    ,PACKEDSY,REALCONST,
                RECORDSY,SETSY,SIGN    ,STRGCONST]+FSYS);;
                ACCEPT(COLON    );
                TYPEDENOTER([SEMICOL    ]+ACCFSYS,[SEMICOL    ]+FSYS);;
                if  CHKSYMSET (([SEMICOL    ]) ) then
                  begin
                      if CHKSYMSET(   (([SEMICOL    ]) )) then
                        begin
                            ACCEPT(SEMICOL    );
                            FIELDLIST(ACCFSYS,FSYS);
                        end
                  end

            end
          else
                if CHKSYMSET(   (([CASESY]) )) then
                  begin
                      VARIANTPT(ACCFSYS,FSYS);
                  end
                else ERRORSET([CASESY,IDENT],'FIELDLIST ')
      end;
procedure VARIANT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure VARIANTPT;

      begin
          TESTSYS([CASESY],FSYS);
          if CHKSYMSET(   (([CASESY]) )) then
            begin
                ACCEPT(CASESY);
                TESTSYS([IDENT],[INTCONST,OFSY,REALCONST,SIGN    ,STRGCONST]+FSYS);;
                ACCEPT(IDENT);
                TESTSYS([COLON    ,OFSY],[IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST]+FSYS);;
                if  CHKSYMSET (([COLON    ]) ) then
                  begin
                      ACCEPT(COLON    );
                      TESTSYS([IDENT],[INTCONST,OFSY,REALCONST,SIGN    ,STRGCONST]+FSYS);;
                      ACCEPT(IDENT);
                      TESTSYS([OFSY],[IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST]+FSYS);
                  end
                      ;
                ACCEPT(OFSY);
                VARIANT([SEMICOL    ]+ACCFSYS,[SEMICOL    ]+FSYS);;
                while  CHKSYMSET (([SEMICOL    ]) ) do
                  begin
                  ACCEPT(SEMICOL    );
                  VARIANT([SEMICOL    ]+ACCFSYS,[SEMICOL    ]+FSYS);
               end

          end
        else ERRORSET([CASESY],'VARIANTPT ')
     end;
procedure CONSTLIST(ACCFSYS,FSYS:SETOFSYS);
   forward;
procedure VARIANT;

     begin
        TESTSYS([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],FSYS);
        if CHKSYMSET(  (([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST]) )) then
          begin
             CONSTLIST([COLON    ],[COLON    ,LPAREN    ,RPAREN    ]+FSYS);;
             ACCEPT(COLON    );
             TESTSYS([LPAREN    ],[RPAREN    ]+FSYS);;
             ACCEPT(LPAREN    );
             TESTSYS([CASESY,IDENT,RPAREN    ],FSYS);;
             if  CHKSYMSET (([CASESY,IDENT]) ) then
               begin
                  FIELDLIST([RPAREN    ],[RPAREN    ]+FSYS);
               end
                  ;
             ACCEPT(RPAREN    );
             TESTSYS(ACCFSYS,FSYS);
          end
        else ERRORSET([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],'VARIANT    ')
     end;
procedure CONSTLIST;

     begin
        TESTSYS([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],FSYS);
        if CHKSYMSET(  (([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST]) )) then
          begin
             CONSTANT([COMA    ]+ACCFSYS,[COMA    ]+FSYS);;
             while  CHKSYMSET (([COMA    ]) ) do
               begin
                  ACCEPT(COMA    );
                  CONSTANT([COMA    ]+ACCFSYS,[COMA    ]+FSYS);
               end

          end
        else ERRORSET([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],'CONSTLIST ')
     end;
procedure SETTYPE;

     begin
        TESTSYS([SETSY],[IDENT,INTCONST,LPAREN    ,OFSY,REALCONST,SIGN    ,STRGCONST]+FSYS);
        ACCEPT(SETSY);
        TESTSYS([OFSY],[IDENT,INTCONST,LPAREN    ,REALCONST,SIGN    ,STRGCONST]+FSYS);;
        ACCEPT(OFSY);
        SIMPLETYPE(ACCFSYS,FSYS);
     end;
procedure FILETYPE;
```

```
      begin
          TESTSYS([FILESY],[IDENT,INTCONST,LPAREN   ,OFSY,REALCONST,SIGN   ,STRGCONST]+FSYS);
          ACCEPT(FILESY);
          TESTSYS([OFSY],[IDENT,INTCONST,LPAREN   ,REALCONST,SIGN   ,STRGCONST]+FSYS);;
          ACCEPT(OFSY);
          SIMPLETYPE(ACCFSYS,FSYS);
      end;
procedure PTRTYPE;

      begin
          TESTSYS([ARROW   ],[IDENT]+FSYS);
          ACCEPT(ARROW   );
          TESTSYS([IDENT],FSYS);;
          ACCEPT(IDENT);
          TESTSYS(ACCFSYS,FSYS);
      end;
procedure VARDEF(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure VARDECPT;

      begin
          TESTSYS([VARSY]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([VARSY]) )) then
                  begin
                      if  CHKSYMSET (([VARSY]) ) then
                        begin
                            if CHKSYMSET(  (([VARSY]) )) then
                              begin
                                  ACCEPT(VARSY);
                                  VARDEF([SEMICOL   ],[SEMICOL   ]+FSYS);;
                                  ACCEPT(SEMICOL   );
                                  TESTSYS([IDENT]+ACCFSYS,FSYS);;
                                  while  CHKSYMSET (([IDENT]) ) do
                                    begin
                                        VARDEF([SEMICOL   ],[SEMICOL   ]+FSYS);;
                                        ACCEPT(SEMICOL   );
                                        TESTSYS([IDENT]+ACCFSYS,FSYS);
                                    end

                              end
                        end

                  end
            end
      end;
procedure VARDEF;

      begin
          TESTSYS([IDENT],[ARRAYSY,ARROW   ,COLON   ,FILESY,INTCONST,LPAREN   ,PACKEDSY,REALCONST,RECORDSY,SETSY,
          SIGN   ,STRGCONST]+FSYS);
          IDLIST([COLON   ],[ARRAYSY,ARROW   ,COLON   ,FILESY,IDENT,INTCONST,LPAREN   ,PACKEDSY,REALCONST,RECORDSY

          SETSY,SIGN   ,STRGCONST]+FSYS);;
          ACCEPT(COLON   );
```

```
     TYPEDENOTER(ACCFSYS,FSYS);
   end;
procedure FNDEC(ACCFSYS,FSYS:SETOFSYS);
  forward;
procedure PROCDEC(ACCFSYS,FSYS:SETOFSYS);
  forward;
procedure PROCFNDECPT;

   begin
     TESTSYS([FUNCSY,PROCSY]+ACCFSYS,FSYS);
     if not CHKSYMSET(ACCFSYS) then
      begin
        if CHKSYMSET(  (([FUNCSY,PROCSY]) )) then
         begin
           while  CHKSYMSET (([FUNCSY,PROCSY]) ) do
            begin
              if CHKSYMSET(  (([PROCSY]) )) then
               begin
                 PROCDEC([FUNCSY,PROCSY]+ACCFSYS,[FUNCSY,PROCSY]+FSYS);
               end
              else
                 if CHKSYMSET(  (([FUNCSY]) )) then
                  begin
                    FNDEC([FUNCSY,PROCSY]+ACCFSYS,[FUNCSY,PROCSY]+FSYS);
                  end
            end

         end
      end
   end;
procedure PROCHEADING(ACCFSYS,FSYS:SETOFSYS);
  forward;
procedure PROCDEC;

   begin
     TESTSYS([PROCSY],FSYS);
     if CHKSYMSET(  (([PROCSY]) )) then
      begin
        PROCHEADING([SEMICOL   ],[SEMICOL   ]+FSYS);;
        ACCEPT(SEMICOL   );
        TESTSYS([BEGINSY,CONSTSY,FUNCSY,IDENT,LABELSY,PROCSY,TYPESY,VARSY],[SEMICOL   ]+FSYS);;
        if CHKSYMSET(  (([IDENT]) )) then
         begin
           ACCEPT(IDENT);
           TESTSYS([SEMICOL   ],FSYS);
         end
        else
           if CHKSYMSET(  (([BEGINSY,CONSTSY,FUNCSY,LABELSY,PROCSY,TYPESY,VARSY]) )) then
            begin
              BLOCK([SEMICOL   ],[SEMICOL   ]+FSYS);
            end
           else ERRORSET([BEGINSY,CONSTSY,FUNCSY,IDENT,LABELSY,PROCSY,TYPESY,VARSY],'PROCDEC   ');
        ACCEPT(SEMICOL   );
        TESTSYS(ACCFSYS,FSYS);
      end
     else ERRORSET([PROCSY],'PROCDEC   ')
```

```
      end;
procedure FNHEADING(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure FNDEC;

      begin
          TESTSYS([FUNCSY],FSYS);
          if CHKSYMSET(  (([FUNCSY]) )) then
            begin
                FNHEADING([SEMICOL   ],[SEMICOL   ]+FSYS);;
                ACCEPT(SEMICOL   );
                TESTSYS([BEGINSY,CONSTSY,FUNCSY,IDENT,LABELSY,PROCSY,TYPESY,VARSY],[SEMICOL   ]+FSYS);;
                if CHKSYMSET(  (([IDENT]) )) then
                  begin
                      ACCEPT(IDENT);
                      TESTSYS([SEMICOL   ],FSYS);
                  end
                else
                      if CHKSYMSET(  (([BEGINSY,CONSTSY,FUNCSY,LABELSY,PROCSY,TYPESY,VARSY]) )) then
                        begin
                            BLOCK([SEMICOL   ],[SEMICOL   ]+FSYS);
                        end
                      else ERRORSET([BEGINSY,CONSTSY,FUNCSY,IDENT,LABELSY,PROCSY,TYPESY,VARSY],'FNDEC     ');
                ACCEPT(SEMICOL   );
                TESTSYS(ACCFSYS,FSYS);
            end
          else ERRORSET([FUNCSY],'FNDEC     ')
      end;
procedure FORMPARLISt(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure PROCHEADING;

      begin
          TESTSYS([PROCSY],FSYS);
          if CHKSYMSET(  (([PROCSY]) )) then
            begin
                ACCEPT(PROCSY);
                TESTSYS([IDENT],FSYS);;
                ACCEPT(IDENT);
                TESTSYS([LPAREN   ]+ACCFSYS,FSYS);;
                if  CHKSYMSET (([LPAREN   ]) ) then
                  begin
                      FORMPARLISt(ACCFSYS,FSYS);
                  end

            end
          else ERRORSET([PROCSY],'PROCHEADIN')
      end;
procedure FNHEADING;

      begin
          TESTSYS([FUNCSY],FSYS);
          if CHKSYMSET(  (([FUNCSY]) )) then
            begin
                ACCEPT(FUNCSY);
                TESTSYS([IDENT],[COLON   ]+FSYS);;
                ACCEPT(IDENT);
                TESTSYS([COLON    ,LPAREN   ],[IDENT]+FSYS);;
                if  CHKSYMSET (([LPAREN   ]) ) then
                  begin
                      FORMPARLISt([COLON    ],[COLON    ,IDENT]+FSYS);
                  end
                      ;
                ACCEPT(COLON    );
                TESTSYS([IDENT],FSYS);;
                ACCEPT(IDENT);
                TESTSYS(ACCFSYS,FSYS);
            end
          else ERRORSET([FUNCSY],'FNHEADING ')
      end;
procedure FORMPARSPEc(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure FORMPARLISt;

      begin
          TESTSYS([LPAREN    ],FSYS);
          if CHKSYMSET(  (([LPAREN    ]) )) then
            begin
                ACCEPT(LPAREN    );
                FORMPARSPEc([RPAREN    ,SEMICOL   ],[RPAREN    ,SEMICOL    ]+FSYS);;
                while  CHKSYMSET (([SEMICOL    ]) ) do
                  begin
                      ACCEPT(SEMICOL    );
                      FORMPARSPEc([RPAREN    ,SEMICOL    ],[RPAREN    ,SEMICOL    ]+FSYS);
                  end
                      ;
                ACCEPT(RPAREN    );
                TESTSYS(ACCFSYS,FSYS);
            end
          else ERRORSET([LPAREN    ],'FORMPARLIS')
      end;
procedure VALVARPARSp(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure FORMPARSPEc;

      begin
          TESTSYS([FUNCSY,IDENT,PROCSY,VARSY],FSYS);
          if CHKSYMSET(  (([IDENT,VARSY]) )) then
            begin
                VALVARPARSp(ACCFSYS,FSYS);
            end
          else
                if CHKSYMSET(  (([PROCSY]) )) then
                  begin
                      PROCHEADINg(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([FUNCSY]) )) then
                        begin
                            FNHEADING(ACCFSYS,FSYS);
                        end
                      else ERRORSET([FUNCSY,IDENT,PROCSY,VARSY],'FORMPARSPE')
      end;
procedure VALVARPARSD;

      begin
          TESTSYS([IDENT,VARSY],FSYS);
          if CHKSYMSET(  (([IDENT,VARSY]) )) then
            begin
                if  CHKSYMSET ((([VARSY]) ) then
                  begin
                      ACCEPT(VARSY)
                  end
                      ;
                IDLIST([COLON    ],[COLON    ,IDENT]+FSYS);;
                ACCEPT(COLON    );
                TESTSYS([IDENT],FSYS);;
                ACCEPT(IDENT);
                TESTSYS(ACCFSYS,FSYS);
            end
          else ERRORSET([IDENT,VARSY],'VALVARPARS')
      end;
procedure EXPLIST(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure VARACCESS(ACCFSYS,FSYS:SETOFSYS);

      begin
          TESTSYS([ARROW    ,LBRAC    ,PERIOD    ]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([ARROW    ,LBRAC    ,PERIOD    ]) )) then
                  begin
                      while  CHKSYMSET (([ARROW    ,LBRAC    ,PERIOD    ]) ) do
                        begin
                            if CHKSYMSET(  (([PERIOD    ]) )) then
                              begin
                                  ACCEPT(PERIOD    );
                                  TESTSYS([IDENT],FSYS);;
                                  ACCEPT(IDENT);
                                  TESTSYS([ARROW    ,LBRAC    ,PERIOD    ]+ACCFSYS,FSYS);
                              end
                            else
                                  if CHKSYMSET(  (([ARROW    ]) )) then
                                    begin
                                        ACCEPT(ARROW    );
                                        TESTSYS([ARROW    ,LBRAC    ,PERIOD    ]+ACCFSYS,FSYS);
                                    end
                                  else
                                        if CHKSYMSET(  (([LBRAC    ]) )) then
                                          begin
                                              ACCEPT(LBRAC    );
                                              EXPLIST([RBRAC    ],[RBRAC    ]+FSYS);;
                                              ACCEPT(RBRAC    );
                                              TESTSYS([ARROW    ,LBRAC    ,PERIOD    ]+ACCFSYS,FSYS);
                                          end
                        end

                  end
            end
      end;
procedure COMPSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STMTPT;

      begin
          TESTSYS([BEGINSY],FSYS);
          COMPSTMT(ACCFSYS,FSYS);
      end;
procedure STMTSEQ(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure COMPSTMT;

      begin
          TESTSYS([BEGINSY],[CASESY,ENDSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,SEMICOL   ,WHILESY,WITHSY]+
          FSYS);
          ACCEPT(BEGINSY);
          STMTSEQ([ENDSY],[ENDSY]+FSYS);;
          ACCEPT(ENDSY);
          TESTSYS(ACCFSYS,FSYS);
      end;
procedure STMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STMTSEQ;

      begin
          TESTSYS([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,SEMICOL   ,WHILESY,WITHSY]+ACCFSYS,
          FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
                if CHKSYMSET(  (([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,SEMICOL   ,WHILESY,
                                  WITHSY]) )) then
                  begin
                      STMT([SEMICOL   ]+ACCFSYS,[SEMICOL   ]+FSYS);;
                      while  CHKSYMSET (([SEMICOL   ]) ) do
                        begin
                            ACCEPT(SEMICOL   );
                            STMT([SEMICOL   ]+ACCFSYS,[SEMICOL   ]+FSYS);
                        end

                  end
            end
      end;
procedure ASSPROSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure GOTOSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STRUCTSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STMT;

      begin
          TESTSYS([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+ACCFSYS,FSYS);
          if not CHKSYMSET(ACCFSYS) then
            begin
```

```
                if CHKSYMSET(  (([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]) ))
                then
                  begin
                      if  CHKSYMSET (([INTCONST]) ) then
                        begin
                            ACCEPT(INTCONST);
                            TESTSYS([COLON    ],FSYS);;
                            ACCEPT(COLON    );
                            TESTSYS([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,REPEATSY,WHILESY,WITHSY]+ACCFSYS,FSYS)

                        end
                            ;
                      if  CHKSYMSET (([BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,REPEATSY,WHILESY,WITHSY]) ) then
                        begin
                            if CHKSYMSET(  (([IDENT]) )) then
                              begin
                                  ASSPROSTMT(ACCFSYS,FSYS);
                              end
                            else
                                  if CHKSYMSET(  (([GOTOSY]) )) then
                                    begin
                                        GOTOSTMT(ACCFSYS,FSYS);
                                    end
                                  else
                                        if CHKSYMSET(  (([BEGINSY,CASESY,FORSY,IFSY,REPEATSY,WHILESY,WITHSY]) ))
                                        then
                                          begin
                                              STRUCTSTMT(ACCFSYS,FSYS);
                                          end
                        end

                  end
            end
      end;
procedure ACTPARLIST(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure EXPRESSION(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure ASSPROSTMT;

      begin
          TESTSYS([IDENT],FSYS);
          if CHKSYMSET(  (([IDENT]) )) then
            begin
                ACCEPT(IDENT);
                TESTSYS([ARROW    ,ASSIGN   ,LBRAC    ,LPAREN   ,PERIOD   ]+ACCFSYS,FSYS);;
                if  CHKSYMSET (([ARROW    ,ASSIGN   ,LBRAC    ,LPAREN   ,PERIOD   ]) ) then
                  begin
                      if CHKSYMSET(  (([ARROW    ,ASSIGN   ,LBRAC    ,PERIOD   ]) )) then
                        begin
                            VARACCESS([ASSIGN   ],[ASSIGN   ,IDENT,INTCONST,LBRAC    ,LPAREN   ,NILSY,NOTSY,
                            REALCONST,SIGN    ,STRGCONST]+FSYS);;
                            ACCEPT(ASSIGN   );
                            EXPRESSION(ACCFSYS,FSYS);
                        end
                      else
                            if CHKSYMSET(  (([LPAREN   ]) )) then
```

```
                              begin
                                  ACTPARLIST(ACCFSYS,FSYS);
                              end
                  end

            end
          else ERRORSET([IDENT],'ASSPROSTMT')
      end;
procedure GOTOSTMT;

      begin
          TESTSYS([GOTOSY],[INTCONST]+FSYS);
          ACCEPT(GOTOSY);
          TESTSYS([INTCONST],FSYS);;
          ACCEPT(INTCONST);
          TESTSYS(ACCFSYS,FSYS);
      end;
procedure CASESTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure IFSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure REPSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure WITHSTMT(ACCFSYS,FSYS:SETOFSYS);
    forward;
procedure STRUCTSTMT;

      begin
          TESTSYS([BEGINSY,CASESY,FORSY,IFSY,REPEATSY,WHILESY,WITHSY],FSYS);
          if CHKSYMSET(  (([BEGINSY]) )) then
            begin
                COMPSTMT(ACCFSYS,FSYS);
            end
          else
                if CHKSYMSET(  (([IFSY]) )) then
                  begin
                      IFSTMT(ACCFSYS,FSYS);
                  end
                else
                      if CHKSYMSET(  (([CASESY]) )) then
                        begin
                            CASESTMT(ACCFSYS,FSYS);
                        end
                      else
                            if CHKSYMSET(  (([FORSY,REPEATSY,WHILESY]) )) then
                              begin
                                  REPSTMT(ACCFSYS,FSYS);
                              end
                            else
                                  if CHKSYMSET(  (([WITHSY]) )) then
                                    begin
                                        WITHSTMT(ACCFSYS,FSYS);
                                    end
                                  else ERRORSET([BEGINSY,CASESY,FORSY,IFSY,REPEATSY,WHILESY,WITHSY],'STRUCTSTMT')
      end;
procedure IFSTMT;
```

```
label
    4;
    begin
    TESTSYS([IFSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,
    REPEATSY,SIGN    ,STRGCONST,THENSY,WHILESY,WITHSY]+FSYS);
    ACCEPT(IFSY);
    EXPRESSION([THENSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,THENSY,WHILESY,WITHSY]+
    FSYS);;
    ACCEPT(THENSY);
    STMT(ACCFSYS,FSYS);
    4 :

          if SYM = ELSESY then begin ACCEPT (ELSESY) ; if SYM =
          IFSY then IFSTMT(ACCFSYS,FSYS) else STMT(ACCFSYS,FSYS); goto 4  end ;
    end;
    procedure CASEBODY(ACCFSYS,FSYS:SETOFSYS);forward;
    procedure CASESTMT;

    begin
    TESTSYS([CASESY],[ENDSY,IDENT,TNTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,OFSY,REALCONST,STGN    ,STRGCONST]+
    FSYS);
    ACCEPT(CASESY);
    EXPRESSION([OFSY],[ENDSY,IDENT,INTCONST,OFSY,REALCONST,SIGN    ,STRGCONST]+FSYS);;
    ACCEPT(OFSY);
    CASEBODY([ENDSY],[ENDSY]+FSYS);;
    ACCEPT(ENDSY);
    TESTSYS(ACCFSYS,FSYS);
    end;
    procedure CASEBODY;

    begin
    TESTSYS([IDENT,INTCONST,REALCONST,SIGN    ,STRGCONST],FSYS); if CHKSYMSET(  (([IDENT,TNTCONST,REALCONST,
                                                                              SIGN    ,STRGCONST]) ))
    then
    begin
    CONSTLIST([COLON    ],[BEGINSY,CASESY,COLON    ,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+
    FSYS);;
    ACCEPT(COLON    );
    STMT([SEMICOL    ]+ACCFSYS,[SEMICOL    ]+FSYS);;
    while  CHKSYMSET (([SEMICOL    ]) ) do
    begin
    ACCEPT(SEMICOL    );
    CONSTLIST([COLON    ],[BEGINSY,CASESY,COLON    ,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+
    FSYS);;
    ACCEPT(COLON    );
    STMT([SEMICOL    ]+ACCFSYS,[SEMICOL    ]+FSYS);
    end

    end  else ERRORSET([IDENT,TNTCONST,REALCONST,SIGN    ,STRGCONST],'CASEBODY  ') end;
    procedure FORSTMT(ACCFSYS,FSYS:SETOFSYS);forward;
    procedure REPEATSTMT(ACCFSYS,FSYS:SETOFSYS);forward;
    procedure WHILESTMT(ACCFSYS,FSYS:SETOFSYS);forward;
    procedure REPSTMT;

    begin
    TESTSYS([FORSY,REPEATSY,WHILESY],FSYS); if CHKSYMSET(  (([WHILESY]) )) then
```

```
begin
WHILESTMT(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([REPEATSY]) )) then
begin
REPEATSTMT(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([FORSY]) )) then
begin
FORSTMT(ACCFSYS,FSYS);
end  else ERRORSET([FORSY,REPEATSY,WHILESY],'REPSTMT    ') end;
procedure WHILESTMT;

begin
TESTSYS([WHILESY],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,
REALCONST,REPEATSY,SIGN    ,STRGCONST,WITHSY]+FSYS);
ACCEPT(WHILESY);
EXPRESSION([DOSY],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+FSYS);;
ACCEPT(DOSY);
STMT(ACCFSYS,FSYS);
end;
procedure REPEATSTMT;

begin
TESTSYS([REPEATSY],[BEGINSY,CASESY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,
REALCONST,SEMICOL    ,SIGN    ,STRGCONST,UNTILSY,WHILESY,WITHSY]+FSYS);
ACCEPT(REPEATSY);
STMTSEQ([UNTILSY],[IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST,UNTILSY]+
FSYS);;
ACCEPT(UNTILSY);
EXPRESSION(ACCFSYS,FSYS);
end;
procedure FORSTMT;

begin
TESTSYS([FORSY],FSYS); if CHKSYMSET(  (([FORSY]) )) then
begin
ACCEPT(FORSY);
TESTSYS([IDENT],[ASSIGN    ,BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IFSY,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY

REALCONST,REPEATSY,SIGN    ,STRGCONST,WHILESY,WITHSY]+FSYS);;
ACCEPT(IDENT);
TESTSYS([ASSIGN    ],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY

REALCONST,REPEATSY,SIGN    ,STRGCONST,WHILESY,WITHSY]+FSYS);;
ACCEPT(ASSIGN    );
EXPRESSION([DOWNTOSY,TOSY],[BEGINSY,CASESY,DOSY,DOWNTOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,LBRAC
LPAREN    ,NILSY,NOTSY,REALCONST,REPEATSY,SIGN    ,STRGCONST,TOSY,WHILESY,WITHSY]+FSYS);;
if CHKSYMSET(  (([TOSY]) )) then
begin
ACCEPT(TOSY)
end  else
if CHKSYMSET(  (([DOWNTOSY]) )) then
begin
ACCEPT(DOWNTOSY)
end  else ERRORSET([DOWNTOSY,TOSY],'FORSTMT    ');
EXPRESSION([DOSY],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+FSYS);;
ACCEPT(DOSY);
```

```
STMT(ACCFSYS,FSYS);
end  else ERRORSET([FORSY],'FORSTMT    ') end;
procedure RECVARLIST(ACCFSYS,FSYS:SETOFSYS);forward;
procedure WITHSTMT;

begin
TESTSYS([WITHSY],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY]+FSYS);
ACCEPT(WITHSY);
RECVARLIST([DOSY],[BEGINSY,CASESY,DOSY,FORSY,GOTOSY,IDENT,IFSY,INTCONST,REPEATSY,WHILESY,WITHSY]+FSYS);;
ACCEPT(DOSY);
STMT(ACCFSYS,FSYS);
end;
procedure RECVARLIST;

begin
TESTSYS([IDENT],FSYS); if CHKSYMSET(  (([IDENT]) )) then
begin
ACCEPT(IDENT);
VARACCESS([COMA    ]+ACCFSYS,[COMA    ]+FSYS);;
while  CHKSYMSET ((([COMA    ]) ) do
begin
ACCEPT(COMA    );
TESTSYS([IDENT],[ARROW    ,LBRAC    ,PERIOD    ]+FSYS);;
ACCEPT(IDENT);
VARACCESS([COMA    ]+ACCFSYS,[COMA    ]+FSYS);
end

end  else ERRORSET([IDENT],'RECVARLIST') end;
procedure SIMPLEEXP(ACCFSYS,FSYS:SETOFSYS);forward;
procedure EXPRESSION;

begin
TESTSYS([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],FSYS); if CHKSYMSET(
                                                                                                ([IDENT,
                                                                                                INTCONST,LBRAC
                                                                                                ,LPAREN    ,
                                                                                                NILSY,NOTSY,
                                                                                                REALCONST,SIGN
                                                                                                ,STRGCONST]) )


then
begin
SIMPLEEXP([EQ    ,INSY,RELOPMEQ    ]+ACCFSYS,[EQ    ,INSY,RELOPMEQ    ]+FSYS);;
while  CHKSYMSET (([EQ    ,INSY,RELOPMEQ    ]) ) do
begin  if CHKSYMSET(  (([EQ    ,INSY,RELOPMEQ    ]) )) then
begin
if CHKSYMSET(  (([EQ    ]) )) then
begin
ACCEPT(EQ    )
end  else
if CHKSYMSET(  (([RELOPMEQ    ]) )) then
begin
ACCEPT(RELOPMEQ    )
end  else
if CHKSYMSET(  (([INSY]) )) then
begin
ACCEPT(INSY)
```

```
end  else ERRORSET([EQ    ,INSY,RELOPMEQ    ],'EXPRESSION');
SIMPLEEXP([EQ    ,INSY,RELOPMEQ    ]+ACCFSYS,[EQ    ,INSY,RELOPMEQ    ]+FSYS);
end  end

end  else ERRORSET([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],'EXPRESSI

) end;
procedure TERM(ACCFSYS,FSYS:SETOFSYS);forward;
procedure SIMPLEEXP;

begin
TESTSYS([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],FSYS); if CHKSYMSET(
                                                                                      (([IDENT,
                                                                                      INTCONST,LBRAC
                                                                                      ,LPAREN    ,
                                                                                      NILSY,NOTSY,
                                                                                      REALCONST,SIGN
                                                                                      ,STRGCONST]) )

then
begin
if  CHKSYMSET (([SIGN    ]) ) then
begin
ACCEPT(SIGN    )
end
;
TERM([ADDOPMS    ,SIGN    ]+ACCFSYS,[ADDOPMS    ,SIGN    ]+FSYS);;
while  CHKSYMSET (([ADDOPMS    ,SIGN    ]) ) do
begin  if CHKSYMSET(  (([ADDOPMS    ,SIGN    ]) )) then
begin
if CHKSYMSET(  (([ADDOPMS    ]) )) then
begin
ACCEPT(ADDOPMS    )
end  else
if CHKSYMSET(  (([SIGN    ]) )) then
begin
ACCEPT(SIGN    )
end  else ERRORSET([ADDOPMS    ,SIGN    ],'SIMPLEEXP ');
TERM([ADDOPMS    ,SIGN    ]+ACCFSYS,[ADDOPMS    ,SIGN    ]+FSYS);
end  end

end  else ERRORSET([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],'SIMPLEEX

) end;
procedure FACTOR(ACCFSYS,FSYS:SETOFSYS);forward;
procedure TERM;

begin
TESTSYS([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,STRGCONST],FSYS); if CHKSYMSET(  (([
                                                                                      IDENT,INTCONST

                                                                                      LBRAC    ,
                                                                                      LPAREN    ,
                                                                                      NILSY,NOTSY,
                                                                                      REALCONST,
                                                                                      STRGCONST]) ))
then
begin
FACTOR([ANDSY,DIVSY,MODSY,MULOP    ,ORSY]+ACCFSYS,[ANDSY,DIVSY,MODSY,MULOP    ,ORSY]+FSYS);;
while  CHKSYMSET (([ANDSY,DIVSY,MODSY,MULOP    ,ORSY]) ) do
```

```
begin  if CHKSYMSET(  (([ANDSY,DIVSY,MODSY,MULOP   ,ORSY]) )) then
begin
if CHKSYMSET(  (([DIVSY]) )) then
begin
ACCEPT(DIVSY)
end  else
if CHKSYMSET(  (([MODSY]) )) then
begin
ACCEPT(MODSY)
end  else
if CHKSYMSET(  (([MULOP   ]) )) then
begin
ACCEPT(MULOP   )
end  else
if CHKSYMSET(  (([ORSY]) )) then
begin
ACCEPT(ORSY)
end  else
if CHKSYMSET(  (([ANDSY]) )) then
begin
ACCEPT(ANDSY)
end  else ERRORSET([ANDSY,DIVSY,MODSY,MULOP   ,ORSY],'TERM     ');
FACTOR([ANDSY,DIVSY,MODSY,MULOP   ,ORSY]+ACCFSYS,[ANDSY,DIVSY,MODSY,MULOP   ,ORSY]+FSYS);
end  end

end  else ERRORSET([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,STRGCONST],'TERM      ') end

procedure SETCONSTR(ACCFSYS,FSYS:SETOFSYS);forward;
procedure FACTOR;

begin
TESTSYS([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,STRGCONST],FSYS); if CHKSYMSET(  (([
                                                                                              IDENT]) ))
then
begin
ACCEPT(IDENT);
if not CHKSYMSET(ACCFSYS) then
begin  if CHKSYMSET(  (([ARROW   ,LBRAC   ,PERIOD   ]) )) then
begin
VARACCESS(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([LPAREN   ]) )) then
begin
ACTPARLIST(ACCFSYS,FSYS);
end  end

end  else
if CHKSYMSET(  (([LPAREN   ]) )) then
begin
ACCEPT(LPAREN   );
EXPRESSION([RPAREN   ],[RPAREN   ]+FSYS);;
ACCEPT(RPAREN   );
TESTSYS(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([NOTSY]) )) then
begin
ACCEPT(NOTSY);
```

```
FACTOR(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([NILSY]) )) then
begin
ACCEPT(NILSY);
TESTSYS(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([LBRAC   ]) )) then
begin
SETCONSTR(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([INTCONST,REALCONST]) )) then
begin
NUMBER(ACCFSYS,FSYS);
end  else
if CHKSYMSET(  (([STRGCONST]) )) then
begin
ACCEPT(STRGCONST);
TESTSYS(ACCFSYS,FSYS);
end  else ERRORSET([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,STRGCONST],'FACTOR    ') end

procedure MEMBDESGN(ACCFSYS,FSYS:SETOFSYS);forward;
procedure SETCONSTR;

begin
TESTSYS([LBRAC   ],FSYS); if CHKSYMSET(  (([LBRAC   ]) )) then
begin
ACCEPT(LBRAC   );
TESTSYS([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,RBRAC   ,REALCONST,SIGN   ,STRGCONST],FSYS);;
if  CHKSYMSET (([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,SIGN   ,STRGCONST]) ) then
begin  if CHKSYMSET(  (([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,SIGN   ,STRGCONST]) ))
then
begin
MEMBDESGN([COMA   ,RBRAC   ],[COMA   ,RBRAC   ]+FSYS);;
while  CHKSYMSET (([COMA   ]) ) do
begin
ACCEPT(COMA   );
MEMBDESGN([COMA   ,RBRAC   ],[COMA   ,RBRAC   ]+FSYS);
end

end  end
;
ACCEPT(RBRAC   );
TESTSYS(ACCFSYS,FSYS);
end  else ERRORSET([LBRAC   ],'SETCONSTR ') end;
procedure MEMBDESGN;

begin
TESTSYS([IDENT,INTCONST,LBRAC   ,LPAREN   ,NILSY,NOTSY,REALCONST,SIGN   ,STRGCONST],FSYS); if CHKSYMSET(
                                                                                          @([IDENT,
                                                                                          INTCONST,LBRAC
                                                                                          ,LPAREN   ,
                                                                                          NILSY,NOTSY,
                                                                                          REALCONST,SIGN
                                                                                          ,STRGCONST]) )


then
begin
```

```
EXPRESSION([TWODOT    ]+ACCFSYS,[TWODOT    ]+FSYS);;
if  CHKSYMSET (([TWODOT    ]) ) then
begin
ACCEPT(TWODOT    );
EXPRESSION(ACCFSYS,FSYS);
end

end  else ERRORSET([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],'MEMBDESG

) end;
procedure ACTUALPARA(ACCFSYS,FSYS:SETOFSYS);


begin
TESTSYS([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],FSYS); if CHKSYMSET(
                                                                                         (([IDENT,
                                                                                         INTCONST,LBRAC
                                                                                         ,LPAREN    ,
                                                                                         NILSY,NOTSY,
                                                                                         REALCONST,SIGN
                                                                                         ,STRGCONST]) )

then
begin
EXPRESSION([COLON    ]+ACCFSYS,[COLON    ]+FSYS);;
if  CHKSYMSET (([COLON    ]) ) then
begin  if CHKSYMSET(  (([COLON    ]) )) then
begin
ACCEPT(COLON    );
EXPRESSION([COLON    ]+ACCFSYS,[COLON    ]+FSYS);;
if  CHKSYMSET (([COLON    ]) ) then
begin
ACCEPT(COLON    );
EXPRESSION(ACCFSYS,FSYS);
end

end  end

end  else ERRORSET([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],'ACTUALPA

) end;
procedure EXPLIST;

begin
TESTSYS([IDENT,INTCONST,LBRAC    ,LPAREN    ,NILSY,NOTSY,REALCONST,SIGN    ,STRGCONST],FSYS); if CHKSYMSET(
                                                                                         (([IDENT,
                                                                                         INTCONST,LBRAC
                                                                                         ,LPAREN    ,
                                                                                         NILSY,NOTSY,
                                                                                         REALCONST,SIGN
                                                                                         ,STRGCONST]) )

then
begin
EXPRESSION([COMA    ]+ACCFSYS,[COMA    ]+FSYS);;
while  CHKSYMSET (([COMA    ]) ) do
begin
ACCEPT(COMA    );
EXPRESSION([COMA    ]+ACCFSYS,[COMA    ]+FSYS);
end
```

```
end  else ERRORSET([IDENT,INTCONST,LBRACK ,LPAREN   ,NILSY,NOTSY,REALCONST,SIGN   ,STRGCONST],'EXPLIST
) end;
procedure ACTPARLIST;

begin
TESTSYS([LPAREN   ],FSYS); if CHKSYMSET(  (([LPAREN   ]) )) then
begin
ACCEPT(LPAREN   );
ACTUALPARA([COMA   ,RPAREN   ],[COMA   ,RPAREN   ]+FSYS);;
while  CHKSYMSET (([COMA   ]) ) do
begin
ACCEPT(COMA   );
ACTUALPARA([COMA   ,RPAREN   ],[COMA   ,RPAREN   ]+FSYS);
end
;
ACCEPT(RPAREN   );
TESTSYS(ACCFSYS,FSYS);
end  else ERRORSET([LPAREN   ],'ACTPARLIST') end;

begin (* main *)
WRITELN(TTY);
CC:=0;LL:=0;CH:=' ';LEXBGN:=0;LEXSIZE:=0;IDULDP:=0;SYM:=ILLEGAL;
ERRINLINE[0]:=0;ERRINLINE[1]:=0;
LASTERRGIVen:=false;PROCERRCURsor:=0;
ERRPRESENT[0]:=false;ERRPRESENT[1]:=false;
LASTLINE:=false;
RECOVERY:=NOPREVATTMpt;NOOFWARNINGs:=0;
BLANKLINE:=false;BUFFINDEX:=0;
PREVPOSITIon[0]:=0;NOOFERRS:=0;
PREVPOSITIon[1]:=0;
for I:= 1 to BUFFLGTH do
begin ERRBUFFER[0][I]:=' ';ERRBUFFER[1][I]:=' '
end;
LINENO:=0;
ATTMPTRECV:=false;
INITSYMNAmes;INITPREVSEts;INITSYPOS;INITIALISE;
LEXANALYSE;
repeat  PROG      ([EOS],[EOS]);
if not EOF(INPUT) then
LEXERROR(25)
until EOF(INPUT);
PROCESSERRor(BUFFINDEX);
if NOOFERRS = 0 then
      begin  if NOOFWARNINGs > 0 then
      WRITELN(TTY,'???  ',NOOFWARNINGs:3,'   WARNINGS');
WRITELN(TTY,'PROGRAM IS SYNTACTICALLY O.K.');
end
else WRITELN(TTY,'???  ',NOOFERRS:3,'  ERRORS AND  ',NOOFWARNINGs:3,'  WARNINGS');
end.
```